

DelegProof: A Machine-Checked Security Analysis of EIP-7702 Delegation

Rong Qian¹, Yu Cheng^{1*}, Lingyu Gao¹, Yuchang Zhang², and Zengli Guo³

¹ Department of Cyberspace Security, Beijing Electronic Science and Technology Institute, Beijing, China rqian@besti.edu.cn, chengyu.cs@zjut.edu.cn, gaolingyu2026@163.com

² Graduate Department of North China Institute of Computer Systems Engineering zhangyuchang26@mails.ucas.ac.cn

³ Faculty of Information Science and Engineering, Ocean University of China, Qingdao 266100, China guozengli@stu.ouc.edu.cn

Abstract. EIP-7702, live on Ethereum since the Pectra upgrade, lets externally owned accounts delegate their execution to arbitrary contract code with a single signature. The consequences are measurable: 63% of observed delegations point to malicious contracts, with \$2.36M in confirmed losses, and ecosystem guidance already warns about cross-chain replay and front-run initialization. What is missing is a formal account of the problem: the 7702 delegation lifecycle has no formal treatment, and none of the recommended mitigations has been machine-verified. We present DelegProof, to our knowledge the first symbolic formal analysis of EIP-7702 authorization semantics composed with the ERC-4337 EntryPoint pipeline. Our Tamarin models are calibrated by reproducing four documented attack classes, and yield ten machine-checked attack patterns—among them the silent failure of the advertised temporary-delegation bundle, storage confusion across re-delegation, and an ERC-1271 substitution attack that reaches into the signature checks of relying contracts—together with the security invariants that do hold. Formalizing the recommended mitigations shows that banning chainId-0 authorizations eliminates cross-chain replay, that an account-bound initialization gate restores init authorization while an unbound gate is provably still bypassable, and that namespaced storage slots make the storage-confusion class structurally inexpressible. All models and proofs are available at <https://github.com/chengyu-cs/delegproof-artifact>.

Keywords: EIP-7702 · Account Abstraction · ERC-4337 · Formal verification · Symbolic analysis · Tamarin · Blockchain security

1 Introduction

Ethereum’s Pectra upgrade (May 2025) activated *EIP-7702*, a transaction type that lets an externally owned account (EOA) sign its authority over to arbitrary

* Corresponding authors: Yu Cheng and Yuchang Zhang.

contract code: a single signature installs a 23-byte delegation designator into the account, after which every execution against that account—by anyone, at any time, on the delegated path—runs the chosen code [6,10]. The mechanism unlocks batching, sponsorship, and session keys for legacy EOAs, and the ecosystem adopted it immediately: within a year, millions of delegation transactions were issued across seven chains [17].

The security consequences have been equally immediate. A measurement study at USENIX Security [17] finds that 63% of observed delegations point to *malicious* contracts and attributes \$2.36M in confirmed losses (with a further \$10.14M exposed), mirroring the CrimeEnjoyer sweep, in which 97% of mass-delegated addresses were routed to attacker-controlled sweeper contracts [33]. The risk is compounded by the account-abstraction stack: since v0.8, the ERC-4337 EntryPoint integrates 7702 natively and executes an *init* call against the delegated account *before* verifying the user operation’s signature [9,5], an ordering that practitioners have already flagged as an attack surface [11,22]. Ecosystem guidance responded quickly: the official Pectra guide, the SEAL framework, and incident response firms all publish warnings and mitigations [11,27,30]. Yet the guidance is entirely *qualitative*: prose warnings, checklists, and reference implementations, with no formal statement of what an attacker can or cannot do, and no verification that any proposed mitigation actually closes the attacks it targets.

The gap. The empirical literature on 7702 abuse is growing quickly [17,24,28], yet the authorization semantics of EIP-7702—singly, and in composition with the ERC-4337 EntryPoint pipeline—have never been subjected to a formal security analysis (we survey venues and preprint servers in Sect. 3). Three questions that decide the safety of every delegated account therefore remain open in any rigorous sense: (i) what exactly can an attacker achieve with one phished authorization signature, across chains and through the EntryPoint pipeline? (ii) which invariants *do* hold in the current protocol, so that wallets and auditors know what they may rely on? and (iii) do the recommended mitigations restore the violated properties, and under which formulation? Empirical work measures the damage [17]; practitioner write-ups speculate about conditions such as cross-chain nonce alignment [27]; and even the formal verification that does exist in the ecosystem (Safe’s 4337 module) explicitly assumes a “well-behaved” EntryPoint, leaving the reference implementation itself unverified [25].

This paper. We answer all three questions with *DelegProof*, a family of Tamarin models [2] that encode the protocol-level semantics of 7702 authorization processing: the authorization-tuple validation chain from the specification, the chainId-0 universal-authorization design, delegation persistence across failed transactions, re-delegation and revocation. The family also encodes the v0.8 EntryPoint execution order (init-before-validate). Model *fidelity* is calibrated in the standard way for protocol analyses: the model reproduces four independently documented attack classes, which it would fail to do if the abstraction dropped their enabling

conditions. On top of the calibrated model we machine-check a systematic property suite, and then re-verify the suite under two mitigation variants.

Our contributions:

1. **A machine-checked formal model** of the EIP-7702 delegation lifecycle composed with the ERC-4337 v0.8 EntryPoint pipeline (Sect. 4), validated by reproducing four known attack classes as machine-checked traces.
2. **Ten machine-checked attack patterns** (Sect. 5): machine-checkable forms of the known classes (cross-chain replay of chainId-0 authorizations, unauthenticated or signature-free EntryPoint phases, single-signature full-pipeline compromise) alongside patterns absent from practitioner literature: a revocation race that invalidates rescue signatures, the silent failure of the temporary-delegation bundle, indirect-delegation bricking, and a symbolic treatment of the ERC-1271 substitution attack, which extends the surface to relying contracts.
3. **Verified security invariants** (Sect. 5): delegation soundness (every delegation traces back to the owner’s signature on that exact message), scoped-signature chain confinement, and the wiring invariant that unauthorized execution requires a prior delegation.
4. **Verified mitigations** (Sect. 6): we formalize the remedies recommended by the ecosystem (banning chainId-0 authorizations [24], gated, signature-checked initialization [11], and namespaced storage slots [14]) and prove the *falsified* $\rightarrow$ *verified* flip for the first two. The naive form of the initialization gate, a signature not bound to the account, is provably still bypassable, a machine-checked confirmation of the EIP-712 domain-separation practice [4,1]. The namespaced-storage fix makes its confusion class structurally inexpressible.
5. **Refined preconditions** (Sect. 5): the model isolates the exact structural condition under which a chainId-0 authorization replays across chains (per-chain nonce alignment), turning the folklore warning that chainId 0 is replayable into a condition wallets can check.
6. **Forward-looking analysis** (Sect. 7): the chainId-0 semantics are propagating to new standards (ERC-7964 cross-chain signature domains, EIP-8130 account configuration), and our verified conclusions apply to this emerging family with explicit, checkable boundary conditions.

All models, proofs, and replication scripts are available at <https://github.com/chengyu-cs/delegproof-artifact>; the full baseline model re-verifies in 502 seconds on a commodity CPU, and the complete model family in under half an hour.

Outline. Sect. 2 introduces the protocol mechanics; Sect. 3 surveys related work; Sect. 4 presents the threat model, abstraction, and Tamarin formalization; Sect. 5 reports the attack and security results; Sect. 6 the verified mitigations; Sect. 7 discusses implications; Sect. 8 the limitations; Sect. 9 concludes.

2 Background: EIP-7702 and the ERC-4337 EntryPoint

2.1 EIP-7702 Authorization Semantics

EIP-7702 [6] extends Ethereum transactions with an *authorization list* whose entries authorize designating the transaction sender (or any account whose key signed the entry) to delegate execution. Each entry carries a chain identifier, a delegate address, a nonce, and an ECDSA signature (y, r, s) by the *authority* (the EOA owner) over a commitment to the triple $(chain\_id, address, nonce)$. Processing an entry validates, in order: the recovered authority’s account must currently hold empty code *or already be delegated*; the nonce must equal the authority’s on-chain nonce; and the chain identifier must match. On success, the account’s code becomes the 23-byte designator `0xef0100 || address`; setting the delegate to zero clears the code (revocation). Four design decisions dominate the security outcome:

1. **ChainId 0 is universal.** An entry with $chain\_id = 0$ validates on *every* chain. The EIP’s rationale frames this as enabling delegation without committing to a specific chain; practitioners flag it as a cross-chain replay hazard [27,30].
2. **Failed entries are skipped, not reverted.** If any check fails, that entry is ignored and the transaction continues. In particular, *a transaction that reverts after processing still leaves the delegation installed*: delegation persistence is independent of transaction success.
3. **Re-delegation requires existing delegation.** A non-empty account accepts a new designator only if it is already delegated (from empty code only the first delegation is possible); combined with the nonce check this gives a linear delegation history per account.
4. **Delegated code is callable by anyone.** Once delegated, the account’s code executes for *any* caller at *any* time; whether that is safe depends entirely on the delegate’s internal access control. The EIP relaxes EIP-3607 [21] so delegated accounts can still originate transactions, preserving the owner’s escape hatch.
5. **Authorization lists process permissively.** Tuples are checked sequentially against the account’s *current* nonce; a tuple that fails any check is skipped without affecting the transaction, and with multiple valid tuples for the same authority the last one wins. This is what makes single-transaction delegate-and-revoke bundles possible; as we show in Sect. 5, it is also what lets such bundles fail silently.

2.2 ERC-4337 EntryPoint Integration

ERC-4337 [5,32] routes account abstraction through an alternative mempool of *user operations* (userOps) settled by a global singleton *EntryPoint* contract. A bundler submits a bundle of userOps to `handleOps`, which for each userOp: (i) creates the sender account if needed, (ii) calls `validateUserOp` (returning a

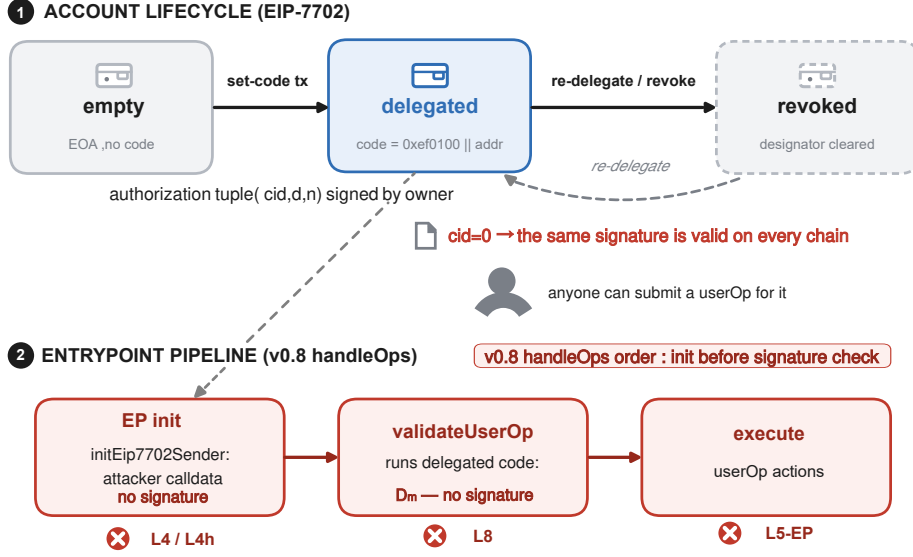


Fig. 1. Analyzed surface: the account lifecycle (top) moves between empty (plain EOA), delegated, and revoked states as the owner signs authorization tuples; a chainId-0 tuple ($cid = 0$) validates on every chain, and anyone may submit a userOp for a delegated account. The v0.8 EntryPoint pipeline (bottom) processes `handleOps` with the initialization phase (`initEip7702Sender`) executing *before* any signature check; the attack-property badges (L4/L4h, L8, L5-EP) mark the phases machine-checked to be reachable without any owner signature (Sect. 5).

validation stake decision), and (iii) executes the operation, optionally calling back for post-operation fees.

Reference implementation v0.8 (March 2025) integrates 7702 natively [9]: a userOp whose `initCode` begins with the `0x7702` marker takes the *7702 sender-creation path*. The EntryPoint first submits an authorization list delegating the sender to the designated contract, then calls `initEip7702Sender`, which forwards the remainder of the init code as *arbitrary calldata to the delegated account*: source comments note it “can be called multiple times.” Only afterwards does `validateUserOp` run, executing *the delegated code* as the validation logic. Two properties matter for our analysis: the `userOpHash` that the (delegate-implemented) validation checks binds the delegate’s address into the signed digest (init-code hash override), and *init runs before any signature verification*. The v0.9 refactor preserves this ordering; we verified by diffing the two releases. Figure 1 summarizes the composed mechanics.

2.3 Known Attack Surface (Qualitative)

Practitioner sources already document fragments of the risk: the official Pectra guide warns that attackers can front-run wallet initialization and recommends

two mitigations, requiring the user’s signature over initialization parameters (the *initWithSig* pattern) and restricting *init* to *EntryPoint*-initiated flows [11]; security teams catalog the same surface qualitatively [22,15,23]; and the phishing literature demonstrates that a single phished authorization can be *remotely re-activated* through *EntryPoint* without any further victim signature [24]. What is missing, and what a symbolic analysis delivers, is a systematic statement of which properties fail, under which preconditions, and which mitigations provably restore them.

3 Related Work

Empirical studies of EIP-7702 abuse. The measurement study of Huang et al. [17] (USENIX Security 2026) quantifies the deployment of 7702 across seven chains, finding 63% of delegations malicious, \$2.36M in confirmed losses and \$10.14M in exposure. Qi et al. [24] document phishing campaigns that harvest authorization signatures and, crucially for our work, show that the *EntryPoint*’s **handleOps** can re-activate an installed malicious delegation *without any new victim signature*; they recommend banning *chainId=0* authorizations or binding context metadata, which is precisely the mitigation family we machine-verify in Sect. 6. Shao et al. [28] study address misuse including 7702-based EOA takeover patterns. These works measure *what happened*; our work establishes *what can happen and when*, at the protocol level, with machine-checked preconditions. The two are complementary, and our models are calibrated against their documented attacks.

Formal analysis of blockchain protocols. Symbolic analysis with Tamarin [2] has an established track record on deployed protocols, from EMV card payments [3] to TLS 1.3 [7] and payment-channel constructions at FC [12]. Closest to our setting, a concurrent preprint formalizes agent *payment* protocols (x402 and successors) in Tamarin [19]; its objects are payment flows, not account authorization, and it does not treat EIP-7702 or the *EntryPoint* pipeline, making the analyses complementary. On the Ethereum side, KEVM provides a complete EVM semantics [16] suited to single-contract functional correctness; our object is the *protocol* that governs authorization lifecycles across chains and pipeline stages, which we abstract at the message level (Sect. 4) following the EMV-style granularity that proved effective for deployed payment protocols [3]. To our knowledge, no symbolic or machine-checked analysis of EIP-7702 exists (we searched ePrint, arXiv, and the major venues through September 2026), and no mitigation for it has been machine-verified. The closest signal is on the integrator side: Safe runs formal verification on its 4337 module, but explicitly under the assumption of a “well-behaved” *EntryPoint*, leaving the reference *EntryPoint* itself unverified, a gap the project tracks in the open [25].

Practitioner guidance and engineering mitigations. The official Pectra guide [11] prescribes the *initWithSig* and *EntryPoint*-only patterns; the SEAL framework

publishes verification checklists for 7702 transactions [27]; SlowMist and Scam Sniffer track incident trends [30,26]; and audited engineering artifacts instantiate the mitigations (Base’s `eip-7702-proxy` realizes signature-gated initialization for Coinbase smart wallets [1]). None of these carries a formal guarantee or states the conditions under which it is sound; DelegProof supplies that verification layer.

4 Security Model and Formalization

4.1 Threat Model

We analyze a Dolev–Yao attacker [8] with full network control (interception, injection, replay of messages and transactions) extended with *on-chain agency*: the attacker may submit arbitrary transactions on any chain, submit arbitrary user operations to the EntryPoint, and invoke any delegated account directly at any time. The Ethereum consensus and the EVM execution semantics are assumed correct; the EntryPoint is honest and matches the reference v0.8 implementation (we analyze its *rules*, not implementation bugs). Signatures are unforgeable in the symbolic sense: without the owner’s secret key, no valid signature can be produced.

Owners hold their keys securely but are phishable: we inject attacker knowledge with authorization signatures that the owner was induced to sign: the empirically dominant infection path [17,24]. Phished *userOp* signatures are likewise attacker knowledge. *Delegate contracts* are instantiated in both honest and malicious forms: an honest delegate (D_h) enforces per-action owner signatures and nonces internally; a malicious delegate (D_m) enforces nothing: the CrimeEnjoyer sweeper profile, which accepts any caller and drains on contact [33]. Anyone may deploy either kind.

4.2 Abstraction and Assumptions

Following the granularity of prior Tamarin analyses of deployed payment protocols [3], we model the *authorization protocol*, not the EVM. Table 1 states the abstraction decisions; each is chosen so that the four calibration attacks of Sect. 4.5 remain expressible. Accounts are symbolic public keys; ECDSA and keccak are symbolic signing and hashing; gas and balances are abstracted away (the attacker cannot be priced out, the conservative direction); the delegate’s internal storage is abstracted into the D_h/D_m behavioral split. Two chains instantiate the multi-chain setting minimally but sufficiently: every cross-chain property we state is a two-chain statement, and replay semantics are pairwise by construction (a `chainId-0` entry validates per-chain independently).

4.3 Tamarin Formalization

The model family is layered: each layer strictly extends the previous and re-establishes inherited lemmas. The stack runs from `min` (single chain, 7 rules /

Table 1. Modeling assumptions (fidelity boundary).

Aspect	Abstraction (rationale)
Accounts / crypto	Public keys; symbolic ECDSA and keccak (standard Dolev–Yao; same level as [3,7])
Gas, fees, balances	Abstracted; attacker never resource-limited (conservative for all stated attacks)
Delegate internals	Behavioral split D_h/D_m ; delegate storage abstracted (access-control behavior is what matters at protocol level)
Chains	Two concrete chains C_1, C_2 ; per-chain states, minimal sufficient instance of multi-chain semantics
Signing oracle	One OwnerSigned event per message: signing-artifact identity, i.e., the attacker possesses <i>signatures</i> , and a repeated event adds no capability; a full control run without this restriction reproduces every baseline verdict (all attack witnesses and invariants unchanged)
Signing channels	Delegation-tuple, userOp, and arbitrary-message signatures carry distinct constructor tags (models typed signing; see Sect. 7.1)
EntryPoint	Honest, v0.8 semantics; bundler untrusted (“anyone may submit”), consistent with the 4337 trust model [32]
Execution layer	Consensus and EVM correct; failed transactions do not revert installed delegation (spec-mandated)

3 lemmas, delegation soundness) through **dual** (two chains, 10/4, replay) and **ep** (the full baseline: 15/7, EntryPoint pipeline) to **revoke** (16/10, revocation semantics, races, indirect delegation). Focused extensions cover the batch semantics (**batch**, 7/4), the ERC-1271 verification paths (**erc1271**, 10/4), and storage-slot persistence (**storage**, 8/3); mitigated variants add **fix1/fix2** (12/8 and 15/8) plus the namespaced-storage variant of Sect. 6. All verdicts quoted in this paper re-verified on logged runs (Tamarin 1.12.0); the full baseline re-verifies in 502 s, and the complete family, including mitigated variants and control runs, verifies in under an hour on one commodity CPU core.

State facts. Persistent facts **Code**(*a*, *s*, *c*) and linear facts **Nonce**(*a*, *n*, *c*) track each account’s code state (empty, or delegated to *d*) and nonce per chain; **OwnerSigned**(*a*, *m*) and **OwnerSignedOp**(*a*, *m*) record owner signing events on the two tagged channels; **Key** holds secret keys. Events **SetCode**, **Revoked**, and **UnauthorizedExec** mark the observable lifecycle steps, and **EPInitExecuted**, **EPValidate**, **EPExecuted** the EntryPoint pipeline steps.

Rule groups. (i) *Authorization processing* implements the EIP’s validation chain as guarded rules, one per (signature channel $\times$ scoped/universal $\times$ empty/delegated) combination: submission of a scoped entry requires $\text{Code}(a, \text{empty}, c) \wedge$

$\text{Nonce}(a, n, c)$ and a valid signature on $\langle\langle c, d \rangle, n\rangle$; universal entries sign $\langle\langle c_0, d \rangle, n\rangle$ and validate on either chain. Re-delegation additionally requires an existing delegation; revocation is a delegation to the zero address (both scoped and universal forms). The skip-on-failure semantics is realized structurally: a rule whose guards fail is simply not applicable, and failed transactions leave **Code** untouched (rule **TxRevert** is a no-op on state). (ii) *Direct execution*: calling a delegated account is unpermissioned (**CallMalicious** fires for any calldata under D_m ; **CallHonest** requires an owner signature, the delegate’s internal check). (iii) *EntryPoint pipeline* fixes the v0.8 order:

```

rule EPIInit:           // phase 1: init call, BEFORE signature check
[ Code(a, deleg(d), c), In(<a, cd>) ]
--[ EPIInitExecuted(a, d, cd, c) ]-> [ Code(a, deleg(d), c) ]
rule EPValidateMal:     // phase 2: delegated validateUserOp
[ Code(a, deleg(Dm), c), In(<a, uop>) ]
--[ EPValidate(a, Dm, c) ]-> [ Validated(a, Dm, c) ]
rule EPEExecute:       // phase 3
[ Validated(a, d, c) ] --[ EPEExecuted(a, d) ]-> [ ]

```

EPIInit fires with *attacker-chosen calldata and no signature of any kind*, exactly the reference behavior (**initEip7702Sender** forwards arbitrary init code to the delegated account and “can be called multiple times” [9]). The honest-validate variant mirrors the direct-call split: under D_h the rule demands an owner’s userOp signature; under D_m it demands nothing.

4.4 Property Suite

We state properties as trace formulas and check them in two modes. *Attack properties* are existentials (or equivalently universals expected to be *falsified*): a Tamarin witness trace is a machine-checked attack. *Security properties* are universals over all traces, proven by backward search (induction where states repeat). Tamarin’s summary prints exists-trace witnesses as *falsified* — *found trace*; throughout the paper we report these as attack witnesses with their trace lengths. The suite:

- L1** *Delegation soundness*: every **SetCode** event for a on chain c is preceded by the owner’s signature on the exact authorization message (channel-scoped).
- L2** *Cross-chain replay* (attack): one chainId-0 signature activates the same delegation on both chains.
- L2s** *Scoped confinement*: a chain-scoped signature for C_1 cannot activate on C_2 .
- L4** *Unauthenticated init* (attack): **EPIInitExecuted** occurs without any owner signature over the calldata (for a malicious delegate); **L4h** states it for an *honest* delegate.
- L8** *Signature-free validation* (attack): **EPValidate** under D_m occurs without any userOp signature.
- L5-EP** *Full-pipeline takeover* (attack witness): **EPEExecuted** under D_m with no new owner signature.
- L-race** *Revocation race* (attack witness): the owner’s revoke signature at nonce n is invalidated by an attacker’s same-nonce delegation winning inclusion.
- L-brick** *Indirect-delegation brick* (attack witness): delegation to a target that is itself delegated, so the 23-byte designator executes as code.

Table 2. Fidelity calibration: independently documented attacks reproduced as machine-checked witnesses.

ID	Documented attack	Source	Repro- duced
A1	Phished delegation $\Rightarrow$ persistent sweeper control	CrimeEnjoyor [33,17]	L2, L5-EP $\checkmark$
A2	chainId-0 replay across chains	EIP rationale; SEAL/SlowMist warnings [6,27,30]	L2 $\checkmark$
A3	EntryPoint remote re-activation without new victim signature	Qi et al. [24]	L8 $\checkmark$
A4	init-before-validate surface; repeatable attacker-calldata init	official guide; v0.8 source [11,9]	L4, L4h $\checkmark$

L-silent-batch *Bundle silent failure* (attack witness): the temporary-delegation bundle’s tuples are all skipped after nonce consumption while the transaction succeeds and the prior delegation survives.

L-1271 *Signature-verification substitution* (attack): under a malicious delegate, an ERC-1271 approval is issued for a message the owner never signed; the undelegated and honest-delegate paths satisfy the corresponding soundness property.

L5’ *Wiring invariant*: unauthorized execution requires a prior `SetCode` on that chain.

4.5 Fidelity Calibration

A symbolic model is only as good as its faithfulness to the deployments it abstracts. Ours reproduces every independently documented attack class in the 7702×4337 setting: had the abstraction dropped an enabling condition, the corresponding witness would not be reachable.

Table 2 summarizes the four calibration classes and their machine-checked counterparts; all four reproduce in the baseline model, and the invariants that *should* hold (L1, L2s) verify alongside, ruling out an over-permissive accident.

5 Attack Analysis

Table 3 summarizes the machine-checked verdicts: ten attack patterns (witness traces; the first seven on the baseline layers `ep/revoke` of Sect. 4, the last three on the focused extension models) and three verified security invariants. We discuss the attacks in the order of the damage path they enable.

5.1 Cross-Chain Replay of Universal Authorizations (L2)

A chainId-0 authorization is a *universal* capability: the same signature validates the delegation on every chain. The witness (2 steps) activates the same delegate

Table 3. Machine-checked results: rows L2–L5’ are verdicts of the baseline layers (`dual/ep/revoke`, two chains; the latter two carry the EntryPoint v0.8 pipeline); rows L-silent-batch, L-1271, and L-storage are verdicts of the corresponding focused extension models. Trace length counts protocol events in the witness; all verdicts from logged Tamarin 1.12.0 runs.

ID	Property	Mode	Verdict	Steps
<i>Attack patterns (witness traces)</i>				
L2	chainId-0 signature activates on both chains	attack	witness	2
L4	EP init calldata unauthenticated (D_m)	attack	witness	8
L4h	EP init calldata unauthenticated (D_h)	attack	witness	8
L8	EP validation without op signature (D_m)	attack	witness	16
L5-EP	full pipeline w/o new owner signature	attack	witness	2
L-race	revoke signature invalidated by race	attack	witness	2
L-brick	designator executed as code	attack	witness	2
L-silent-batch	temp-delegation bundle silently no-ops, tx succeeds	attack	witness	2
L-1271	ERC-1271 approval without owner signature (D_m)	attack	witness	8
L-storage	address-typed slot consumed as amount view	attack	witness	2
<i>Security invariants (verified)</i>				
L1	delegation soundness (exact-message signature)	security	verified	41
L2s	scoped signatures confined to their chain	security	verified	36
L5’	unauthorized exec needs prior set-code	security	verified	16

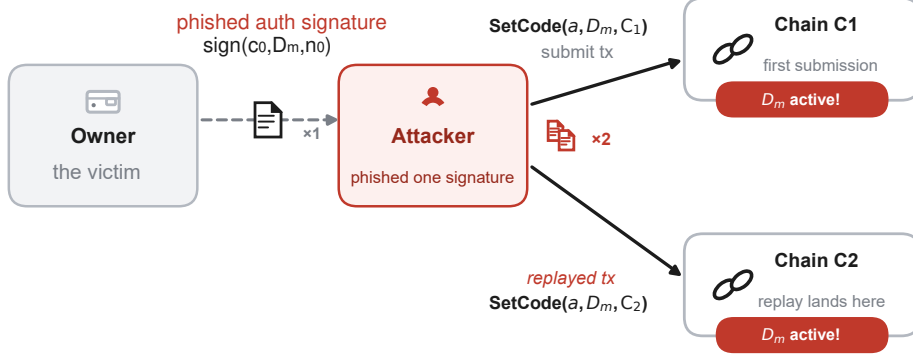


Fig. 2. The cross-chain replay (L2): one phished authorization signature is enough for the attacker to submit the `SetCode` transaction on Chain C_1 and replay it on Chain C_2 : the chainId-0 universal-authorization design validates the same tuple on every chain where the account nonce matches, so the victim’s account ends up with the same malicious delegate D_m active on both chains.

on C_1 and C_2 from one phished signature (Figure 2). The refinement the practitioner literature lacks is the *precondition*: replay requires the account’s nonce to be *aligned* across the two chains at signing time. In the model this is structural (every authorization rule carries the per-chain `Nonce` premise, mirroring the EIP’s on-chain nonce equality check), so a chainId-0 signature replays on exactly the chains where the nonce still matches the signed value. Wallets can turn this into a deployment-time check: *an account whose nonces have diverged across chains is no longer replayable at its old nonce*, and conversely, freshly-created accounts (nonce 0 everywhere) are maximally exposed. This upgrades the folklore warning “chainId 0 is replayable” [27,30] into a concrete, checkable characterization, which follows directly from the guard structure of every authorization rule: *universal signatures replay precisely on the chain set where per-chain nonces equal the signed nonce*.

The complementary invariant **L2s** verifies: a chain-scoped signature cannot fire on a foreign chain (36 proof steps), confirming that the exposure is caused by the universal-authorization design, not by the model’s handling of chains.

5.2 Unauthenticated EntryPoint Initialization (L4, L4h)

In the v0.8 pipeline, `initEip7702Sender` executes attacker-chosen calldata at the delegated account *before* any signature verification. The witness for **L4** (8 steps) has the attacker drive arbitrary initialization logic under a malicious delegate; the practitioner record of this surface [11,22] is thereby given a machine-checked form. The paired witness **L4h** (8 steps) is the sharper statement: the same unauthenticated init executes under an *honest* delegate. The absence of protection is thus at the *protocol* level—it does not depend on the delegate be-

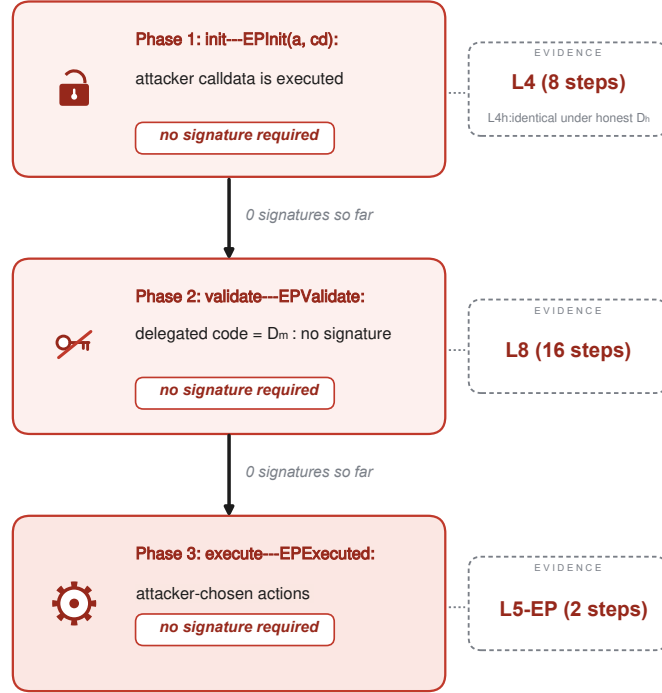


Fig. 3. The EntryPoint pipeline driven end to end without any new owner signature once a malicious delegation is installed: Phase 1 (init) executes attacker-supplied call-data with no signature (L4, 8 steps); Phase 2 (validate) invokes the delegated code D_m , which accepts without a signature (L8, 16 steps); Phase 3 (execute) performs attacker-chosen actions (L5-EP, 2 steps). The evidence column lists the witness identifier and trace length for each phase.

ing malicious—and any mitigation must act at the EntryPoint or signature layer (Sect. 6).

5.3 Signature-Free Validation and Full-Pipeline Takeover (L8, L5-EP)

Under a malicious delegate, `validateUserOp` is the delegate’s code, so nothing prevents it from accepting any submitted `userOp`. The **L8** witness (16 steps) completes validation without any owner signature; chaining into execution, the **L5-EP** witness (2 steps) reaches `EPExecuted` with no new owner action (Figure 3). This machine-checks the remote re-activation reported empirically by Qi et al. [24]: once a phished delegation is installed, the attacker’s control is composable through the standard account-abstraction path, with the EntryPoint itself providing the repeated trigger surface.

5.4 Revocation Race (L-race)

Delegated accounts remain full EOAs: the rescue path is a set-code transaction clearing the designator, signed at the account’s current nonce. But that nonce is a *shared resource*: the **L-race** witness (2 steps) shows an attacker who obtains (or front-runs) any same-nonce delegation, e.g., a second phished authorization, and gets it included first. The nonce increments, the owner’s pre-signed rescue signature is permanently invalidated, and the attacker controls the next delegation step. The practical consequence for incident response: a pre-signed rescue transaction is not a safety net while the attacker holds any usable authorization at the same nonce; revocation must be executed at a nonce the attacker cannot consume.

5.5 Silent Failure of the Temporary-Delegation Bundle (L-silent-batch)

The flagship 7702 user experience is the *temporary delegation* bundle: one transaction whose authorization list carries [delegate-to- D_h @ n , revoke @ $n+1$], upgrading the account for exactly one transaction. The specification’s list semantics (each tuple checked against the account’s *current* nonce, failures skipped silently, the transaction itself unaffected) compose badly with the race above. The **L-silent-batch** witness shows the failure mode: an attacker holding two phished authorizations at consecutive nonces n and $n+1$ (the same precondition as L-race) lets the owner’s bundle confirm *after* consuming both nonces; every tuple in the list then fails its nonce check and is skipped, the transaction executes and is reported as successful, and any prior delegation, for example an installed malicious one, survives untouched. We machine-check both sides: the bundle performs its round trip when fresh (witness), and the silent-failure trace exists in which the same account keeps its malicious delegation at the moment the bundle transaction succeeds while the owner had signed the bundled revoke (witness, cross-confirmed by a variant witness that forces the two attacker nonce consumptions into the trace explicitly). The practical consequence is sharper than L-race: the victim’s own wallet reports success for a rescue that silently did nothing. The defense surface is correspondingly different: wallets must surface per-tuple application results rather than transaction-level success, a direction the draft ERC-7779 interoperable delegated-account standard takes up [20].

5.6 Indirect Delegation Bricks the Account (L-brick)

The EIP permits delegation to *any* address, including one that is itself delegated. The **L-brick** witness (2 steps) delegates account a_1 to target a_2 where a_2 already carries a designator: the installed “code” for a_1 is now the 23-byte designator of a_2 , whose leading byte `0xef` is the `INVALID` opcode: every inbound call to a_1 reverts. The account is bricked for composability (denial of service), though the owner retains the EIP-3607-relaxed ability to originate a repairing set-code transaction [21]. Two aspects are, to our knowledge, undiscussed in the

practitioner literature: the pattern requires no attacker sophistication beyond choosing a delegated target, and its uglier variant is an *upgrade rug*: a target contract whose controller later delegates *itself* silently changes the effective semantics for every account that delegated to it, a transitive control amplification we return to in Sect. 7.

5.7 Breaking the Signature Verification of Relying Contracts (L-1271)

Delegation does not only endanger the delegated account: it breaks assumptions *other* protocols make about it. A relying contract that verifies a user signature falls back to ERC-1271 `isValidSignature` when the signer has code; after 7702, an EOA can have code at any moment, so the fallback now executes the *delegate*’s verification logic. This exact hazard is recorded as an audit finding (OpenZeppelin’s `isValidSignatureNow` breaks for delegated EOAs [29]) and falls within the contract-targeted attack category that the USENIX measurement attributes a \$10.14M exposure to [17]. Our model adds the three verification paths (erecover for undelegated accounts, delegate-mediated approval for delegated accounts, either channel for the owner’s signing oracle) with the message channel type-separated from the authorization channel. The verdicts form a three-part asymmetry: on the undelegated path, approval implies the owner signed the message (**verified**, 8 steps); under a malicious delegate, an approval is issued for a message the owner never signed (**attack witness**, 8 steps); under an honest delegate that internally demands the owner’s signature, the implication is restored (**verified**, 25 steps). Delegation silently converts every signature check that trusts ERC-1271 into a check of the delegate’s choosing; to our knowledge this is also the first symbolic treatment of ERC-1271 verification itself, and the standards track is already responding (ERC-7739 typed signatures for smart accounts) [31]. The same assumption-breakage class includes `tx.origin` guards and EIP-712 caching schemes, which we treat qualitatively in Sect. 7.

5.8 Storage Confusion Across Re-Delegation (L-storage)

Delegating does not clear the account’s storage, and the official guide warns that residual data from a previous delegate, read under a different implementation’s layout, “can cause unintended behavior” [11]; the engineering mitigation is standardized as the ERC-7201 namespaced-storage formula [14]. We machine-check the documented instance: implementation *A* stores the owner identity at slot 0; after re-delegation, implementation *B* reads slot 0 as a balance. The **L-storage** witness fires the amount view over a value that was authored as an *address*: the guide’s “absurdly large balance” made machine-checked. The property is deliberately narrow (one slot, two typed views) because value confusion is an interpretation-level phenomenon. Sect. 6 shows that the namespaced-storage variant makes such cross-delegation reads inexpressible, giving ERC-7201’s pattern a verification-layer statement for the 7702 setting.

Table 4. Mitigation verification: baseline verdicts flip to *verified* under the corrected fixes; the naive unbound gate remains attackable.

Mitigation	Formalization	Baseline	Mitigated
Fix 1: ban chainId 0	remove universal-authorization rules	L2 witness (attack)	verified (8)
scoped confinement preserved	rules unchanged otherwise	L2s verified	verified (32)
Fix 2 (naive): init requires <i>an</i> owner signature, unbound	signature premise on EPInit , signer not bound to account	L4 witness (attack)	<i>still attack</i> (8)
Fix 2: init requires the account owner’s signature, <i>bound</i>	signature premise with identity binding <i>verifier = account</i>	L4 witness (attack)	verified (6)
Fix 3: namespaced storage keys (ERC-7201)	slot keys prefixed with delegate identity; readers match own namespace only	L-storage witness (attack)	<i>inexpressible by construction</i>

5.9 Verified Invariants (L1, L5’)

Three universal properties verify, giving deployers positive guarantees. **L1** (41 steps): every delegation event traces back to the owner’s signature on exactly that authorization message, so no delegation can appear from forged or reused signatures across channels. **L2s** (36 steps): scoped signatures are chain-confined. **L5’** (16 steps) confirms the wiring invariant: unauthorized execution under D_m is only possible through a prior **SetCode** on that chain. Together they delimit the attack surface: compromise paths flow through signatures the owner actually produced. How far one phished signature reaches is the subject of the next section.

6 Verified Mitigations

The ecosystem’s recommended defenses exist as prose and code but not as proofs. We formalize them and machine-check whether they restore the violated properties; a naive formulation of the second turns out not to.

6.1 Fix 1: Banning Universal Authorizations

Qi et al. [24] recommend prohibiting chainId-0 authorization processing, and wallets increasingly default to scoped signatures. We formalize the fix minimally: the three universal-authorization rules (delegation, re-delegation, revocation) are removed; everything else, including all attack lemmas, is unchanged. Under Fix 1, the replay property **no_crosschain_replay_fixed** (*no* signature

activates delegations on both chains) **verifies** in 8 steps, and scoped confinement L2s re-verifies (32 steps). A full re-verification of the model under Fix 1 (178s) confirms every inherited verdict: the attack witnesses that Fix 1 does not target (L4, L8, L5-EP) are undisturbed, and delegation soundness holds as before. The flip confirms that the cross-chain exposure flows through the universal-authorization rules alone.

Two corollaries follow. First, the revocation channel is part of the fix’s scope: universal *revoke* signatures are equally replayable, which matters for wallets signing blanket revocations. Second, the fix is *local*: since the scoped path already existed, banning chainId 0 removes a capability without touching any honest flow, the formal counterpart of its low deployment cost, and a verification of the recommendation as stated in [24,27].

6.2 Fix 2: Gated Initialization and Why the Naive Gate Fails

The official guidance for the init surface recommends requiring “the user’s signature over initialization parameters” before init calldata executes [11]: the *initWithSig* pattern, as engineered in audited proxies [1]. The design space hides a sharp edge that our model captures exactly.

Naive gate (still attackable). Add to **EPInit** the premise that *some* owner’s signature over the calldata exists, without binding the signing key to the target account. The L4 property *remains falsifiable* (witness, 8 steps): any account owner’s signature authorizes initialization of *any other* delegated account. In a network of delegated accounts, attacker chatter alone, or a signature the attacker legitimately holds for their own account, satisfies the gate. The mitigation as intuitively stated (“require a signature”) is vacuous; a witness proves it.

Bound gate (verified). Bind the gate: **EPInit** now requires $verify(\sigma, calldata) = a$ where a is the account being initialized: the owner of *that* account must have signed *that* calldata. The previously-falsified property **init_requires_owner_auth_fixed** (every init event under D_m is preceded by the target account owner’s signature on the calldata, on either signing channel) now **verifies** in 6 steps.

The three-part statement. Together with the baseline witness, this yields a machine-checked three-part argument: (1) the protocol as deployed offers no init authorization (L4 witness); (2) a signature gate without subject binding does not close it (naive-Fix-2 witness); (3) account-bound gating restores it (Fix-2 verified). In other words, domain separation is a necessary ingredient of the mitigation, the practice the ecosystem codified in EIP-712’s domain separators [4], here verified against a concrete bypass. Audited engineering variants that bind the account and initialization payload into the owner’s signature (Base’s **eip-7702-proxy** for Coinbase smart wallets [1]) are instances of the verified pattern. Under Fix 2, the inherited invariants re-verify (delegation soundness, 41 steps; scoped confinement, 36 steps), while the attacks Fix 2 does not target (cross-chain replay, signature-free validation, full-pipeline takeover) remain reachable: the fix is scoped to the initialization surface.

6.3 Fix 3: Namespaced Storage Slots

For the storage-confusion class (L-storage), the engineering mitigation is standardized: ERC-7201’s namespaced-storage formula, recommended for 7702-ready wallets by the draft ERC-7779 [14,20]. We model the pattern by prefixing every storage key with the delegate identity: writes land in slots keyed by $\langle\langle D, s \rangle\rangle$ and a reader under D can only match its own namespace. Under this variant a cross-delegation read is inexpressible: the reader’s premise cannot match another delegate’s write, since the slot terms have distinct constructor heads: the same guard-level argument as the nonce precondition, and analogous to the channel tagging of Sect. 7.1. The same-namespace round trip still works (machine-checked witness in the namespaced model), so the fix removes the cross-delegation reads and nothing else.

6.4 Scope of the Fixes

The fixes are protocol-level and target the properties they claim: Fix 1 closes cross-chain replay; Fix 2 closes unauthenticated init. Neither is a cure for L5/L8-class compromise under an installed *malicious* delegation: once the owner has delegated to attacker code, no protocol rule can distinguish the attacker’s use from the owner’s intent; that boundary is inherent (the delegation *is* the authorization), and the corresponding defenses are wallet-side (signature blindness mitigations, scoped defaults, revocation-at-safe-nonce procedures), consistent with the guidance in [11,27]. Formalizing the boundary itself is useful: it tells standard authors exactly which layer must carry each remaining defense.

7 Discussion

7.1 Cross-Channel Signature Confusion via Blind Signing

A modeling finding deserves separate nameable status. Our first formulation of the userOp signing channel allowed a delegation-tuple signature and a userOp signature to share a shape; Tamarin’s attacker instantly unified them, and delegation became possible with signatures the owner never intended as authorizations. Far from being an artifact, this is the *real* phishing vector: wallets that blind-sign structured payloads hand the attacker exactly the signature objects our accidental unification simulated, and incident reports confirm that phishing pages elicit 7702 authorizations disguised as routine signatures [24,17]. We repaired the model with distinct op-tags per signing channel (Table 1), which is also the correct deployment prescription: *typed, channel-separated signing is load-bearing for 7702 safety*, and our L1 soundness invariant is verified *under* the separation. The machine-checked content is that without the tag separation, no invariant of L1’s shape is even statable. Two further instances arose while modeling the ERC-1271 paths of Sect. 5: an untyped message oracle let the attacker mint authorization signatures through it (every invariant fell, including delegation soundness), and a pair-shaped pseudo-tag was unified away;

the counterexample trace showed the oracle’s output matching an authorization tuple verbatim. Only a distinct constructor head restores the separation. Three independent instances suggest the lesson carries beyond our model: typed, channel-separated signing is load-bearing for any 7702-adjacent formal model, and likely for wallets.

7.2 Delegation Breaks Ambient Assumptions

L-1271 is one member of a broader class: once EOAs can carry code, every protocol that conditioned on “an EOA has no code” inherits a new attack surface. The empirical study quantifies the category (\$10.14M exposure across contract-targeted attacks, including `tx.origin` guards and `extcodesize` checks) [17]; audit findings document the ERC-1271 instance [29]; EIP-712 signature-caching schemes fail under persistent delegation for the same reason. Our L-1271 result, to our knowledge, gives the class its first machine-checked instance, and the pattern generalizes: any relying protocol whose verification consults account state must now ask *whose* logic answers. The standards track is responding (ERC-7739 typed signatures for smart accounts, ERC-7779 interoperable delegated accounts) [31,20]; our models are small enough to serve as the verification layer for such proposals.

7.3 Transitive Control: The Upgrade Rug

The L-brick attack’s sharper variant deserves emphasis beyond its DoS reading (Sect. 5): if a widely-delegated target contract later delegates *itself* (for example, its controller installs new logic via 7702), every account that delegated to it silently changes execution semantics. Delegation chains make contract control *transitive* at one signature’s notice, a rug vector aimed at *delegators*, not at the target itself. No protocol rule prevents it; the model surfaces the pattern because it is expressible in the lifecycle, and we flag registry-style mitigations (binding delegation to a codehash commitment) as a direction the formalization could verify.

7.4 The chainId-0 Family: ERC-7964 and EIP-8130

The universal-authorization semantics are propagating. ERC-7964 defines cross-chain EIP-712 signature domains that self-describe as following “the chainId-0 pattern established by EIP-7702” [13], and EIP-8130 builds directly on 7702 delegation designators [18]. Our results transfer with explicit boundary conditions: Fix 1’s verification shows exactly which rule family must be absent for cross-chain replay to be impossible; the nonce-alignment precondition of Sect. 5 characterizes the exposure of any standard that reuses chainId-0 processing; and the Fix-2 three-part argument applies verbatim to any init-like surface in those designs. Standards-track proposals that reuse 7702 semantics inherit a *verified* analysis of its sharpest edge, the analysis we recommend they run before deployment, and which our models make a checklist rather than a research project.

8 Limitations and Future Work

Our analysis is symbolic and protocol-level. Concretely: *(i)* EVM storage semantics inside delegates are abstracted into the honest/malicious behavioral split; attacks that depend on specific storage layouts (e.g., subtle reentrancy) are out of scope, though the protocol-level attack surface we characterize is precisely what storage-level bugs compound with. *(ii)* Gas, mempool economics, and bundler competition are abstracted; the L-race result states the capability (same-nonce invalidations), not the probability of winning inclusion races, which is an economic question. *(iii)* Two chains instantiate multi-chain semantics minimally; all cross-chain statements are pairwise, and we did not model chain forks or replay across chains with identical chain identifiers. *(iv)* The single-signing restriction (Table 1) identifies signing artifacts, not signing events; repeated events add no attacker capability in the symbolic model, but a probabilistic analysis might diverge. Cross-validation against a mechanical semantics (KEVM [16] or Kontrol) for a sample of the witnesses is the natural next step, as is extending the property suite to the paymaster and aggregator roles that we, like the 7702 literature, currently leave out. None of these limitations affects the two verified flips of Sect. 6, whose statements are about rule presence and absence, not about execution detail.

9 Conclusion

EIP-7702 changed Ethereum’s account model with one signature’s worth of authority, and the ecosystem has been paying for that power in phished authorizations ever since. We delivered the missing verification layer: a machine-checked model of the delegation lifecycle composed with the ERC-4337 Entry-Point pipeline, calibrated against documented attacks, that characterizes ten attack patterns with their preconditions, verifies the invariants deployers can rely on, and upgrades the recommended mitigations from prose to proofs, including a verified refutation of the naive form of one of them. As the chainId-0 semantics propagate to new standards, the analysis is designed to travel: the models are small and rerunnable, turning “is this pattern safe?” from folklore into a checkable question.

References

1. Base (Coinbase): eip-7702-proxy: an audited ERC-1967 proxy for EOA upgrades to CoinbaseSmartWallet. <https://github.com/base/eip-7702-proxy> (2025)
2. Basin, D., Cremers, C., Dreier, J., Meier, S., Sasse, R., Schmidt, B.: The Tamarin prover for the symbolic analysis of security protocols. Lecture Notes in Computer Science (LNCS 12741) (2021)
3. Basin, D., Sasse, R., Toro-Pozo, J.: The EMV standard: Break, fix, verify. In: Proc. IEEE Symposium on Security and Privacy (S&P). pp. 1766–1781 (2021)

4. Buterin, V., Vogelsteller, F., Morriess, W.: EIP-712: Typed structured data hashing and signing. Ethereum Improvement Proposal 712 (Final) (2018), <https://eips.ethereum.org/EIPS/eip-712>
5. Buterin, V., Weiss, Y., Tirosh, D., Nacson, S., Forshtat, A., Gazso, K., Hess, T.: ERC-4337: Account abstraction using alt mempool. Ethereum Request for Comments 4337 (2021), <https://eips.ethereum.org/EIPS/eip-4337>
6. Buterin, V., Wilson, S., Garnett, M., Dietrichs, A., Morriess, W.: EIP-7702: Set code for EOAs. Ethereum Improvement Proposal 7702 (Final) (2024), <https://eips.ethereum.org/EIPS/eip-7702>
7. Cremers, C., Horvat, M., Hoyland, J., Scott, S., van der Merwe, T.: A comprehensive symbolic analysis of TLS 1.3. In: Proc. ACM CCS (2017)
8. Dolev, D., Yao, A.C.C.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2), 198–207 (1983)
9. eth-infinitism: account-abstraction: ERC-4337 reference implementation, v0.8.0. <https://github.com/eth-infinitism/account-abstraction> (2025)
10. Ethereum Foundation: Pectra (prague + electra) network upgrade. <https://ethereum.org/en/roadmap/pectra> (2025)
11. ethereum.org: EIP-7702: Empowering EOAs. <https://ethereum.org/en/roadmap/pectra/7702> (2025)
12. Fabianski, G., Stefanski, R., Litos, O.S.T.: A formally verified lightning network. In: Proc. Financial Cryptography and Data Security (FC) (2025)
13. García, E.: ERC-7964: Crosschain EIP-712 signatures. Ethereum Request for Comments 7964 (Draft) (2026), <https://eips.ethereum.org/EIPS/eip-7964>
14. Giordano, F., Croubois, H., García, E., Lau, E.: ERC-7201: Namespaced storage layout. Ethereum Request for Comments 7201 (Final) (2023), <https://eips.ethereum.org/EIPS/eip-7201>
15. Halborn: EIP 7702 security considerations. <https://halborn.com/blog/post/eip-7702-security-considerations> (2025)
16. Hildenbrandt, E., Raghunathan, N., Genc, E., Park, D., Zhang, Y., Tefas, A., Roşu, G.: KEVM: A complete formal semantics of the Ethereum virtual machine. In: Proc. IEEE CSF (2018)
17. Huang, M., Liu, H., Yang, S., Wu, D., Wang, S.: Revealing the dark side of smart accounts: An empirical study of EIP-7702 incurred risks in blockchain ecosystem. In: Proc. USENIX Security Symposium (2026)
18. Hunter, C.: EIP-8130: Account abstraction by account configuration. Ethereum Improvement Proposal 8130 (Draft) (2025), <https://eips.ethereum.org/EIPS/eip-8130>
19. Jiang, K., Yu, M., Chang, Y., Jangid, M.K., Niu, J., Wang, C., Zhang, Y.: A formal analysis of agent payment protocols. arXiv preprint arXiv:2609.00060 (2026)
20. Kim, D., Meissner, R., Patel, A., Kim, J., Fangting, Weiss, Y.: ERC-7779: Interoperable delegated accounts. Ethereum Request for Comments 7779 (Draft) (2024), <https://eips.ethereum.org/EIPS/eip-7779>
21. Lundfall, M.: EIP-3607: Reject transactions from senders with deployed code. Ethereum Improvement Proposal 3607 (Final) (2021), <https://eips.ethereum.org/EIPS/eip-3607>
22. Nethermind: EIP-7702 attack surfaces: What developers should know. <https://www.nethermind.io/blog/eip-7702-attack-surfaces-what-developers-should-know> (2025)
23. OpenZeppelin: The notorious bug digest #5: Post EIP-7702 pitfalls, JIT penalty rebates, and manipulation of recursive functions. <https://www.openzeppelin.com/news/the-notorious-bug-digest-5> (2025)

24. Qi, M., Wang, Q., Li, R., Zhu, T., Chen, S.: EIP-7702 phishing attack. arXiv preprint arXiv:2512.12174 (2025)
25. Safe Foundation: Use reference EntryPoint in formal verification (issue #235). <https://github.com/safe-fndn/safe-modules/issues/235> (2024)
26. Scam Sniffer: Crypto phishing reports (2025). <https://scamsniffer.io> (2025)
27. Security Alliance (SEAL): Verifying EIP-7702 transactions. <https://frameworks.securityalliance.org> (2025)
28. Shao, Z., Zhang, J., Li, Z., Wu, D., Chen, C., Shen, Y., Bao, L., Wang, Y., Chen, J.: Lost in blockchain address misuse: Hidden cross-platform risks and their security impact. In: Proc. USENIX Security Symposium (2026)
29. Sherlock audit contest (Symbiotic Relay): Signature checking would break under EIP-7702 because of `isvalidsignaturenow()` logic (issue #288). <https://github.com/sherlock-audit/2025-06-symbiotic-relay-judging/issues/288> (2025)
30. SlowMist (Cosine): Advisories on EIP-7702 abuse by theft gangs. X security advisories, May–June 2025 (2025)
31. vectorized, Lee, S., Giordano, F., Croubois, H., García, E., Juno, howydev, Atarpara, Oxcuriousapple: ERC-7739: Readable typed signatures for smart accounts. Ethereum Request for Comments 7739 (Draft) (2024), <https://eips.ethereum.org/EIPS/eip-7739>
32. Weiss, Y., Tirosch, D., Forshtat, A., Nacson, S.: ERC-7562: Account abstraction validation scope rules. Ethereum Request for Comments 7562 (2023), <https://eips.ethereum.org/EIPS/eip-7562>
33. Wintermute Research: Crimeenjoyor: EIP-7702 delegation abuse on Ethereum. Public dashboard and advisories, May–June 2025; reported by CoinDesk, Coin-telegraph, and Protos, <https://protos.com/48-of-ethereum-eip-7702-uses-linked-to-crime-says-wintermute/> (2025)