Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Boundless

RISC Zero Solana Verifier

Veridise Inc.

October 6, 2025

► **Prepared For:**

Boundless
https://boundless.network/

► **Prepared By:**

Evgeniy Shishkin
Mark Anthony

► **Contact Us:**

contact@veridise.com

► **Version History:**

October 17, 2025 V2
October 14, 2025 V1

Contents

1 Executive Summary

From Oct. 6, 2025 to Oct. 7, 2025, Boundless engaged Veridise to conduct a security assessment of the RISC Zero Solana Verifier. Veridise conducted the assessment over 4 person-days, with 2 security analysts reviewing the project over 2 days on commit `3be7250`. The review strategy entailed a comprehensive manual examination of the project's source code.

The Groth16 verifier component of this codebase was previously reviewed by Veridise*. Relative to the prior version, the new codebase has been extended with a Router component for directing proof verification requests to the appropriate verifier, together with refactoring of the Verifier library.

Project Summary. The RISC Zero Solana Verifier is a management contract designed to provision multiple Groth16 verifiers, each bound to a specific Risc0 verifier circuit. Because verification circuits may evolve over time, the protocol requires a canonical source of truth where the correct circuit version is coupled with its corresponding Groth16 verifier. This ensures consistency and reliability when proofs are validated. In addition, if a verifier is found to be vulnerable, its corresponding entry can be disabled in the manager contract, preventing users from relying on an insecure version.

Beyond the management functionality, the project incorporates a Groth16 prover library implementation aligned with up-to-date verification circuits.

Specifically, the RISC Zero Solana Verifier provides the following key functions:

- **Register Verifiers:** Add a new Groth16 verifier associated with a unique *Selector* - a 4-byte identifier uniquely representing a set of Risc0 circuits responsible for proof generation. This operation is available only to the contract's owner.
- **Proof Verification:** The primary entry point for users to verify proofs. Forwards requests to the appropriate prover version, provided it has not been disabled.
- **Emergency Disable (Owner-Controlled):** The contract owner can disable a verifier associated with a specific selector.
- **Emergency Disable (Proof-Based):** Any user may disable a verifier if they can provide a valid Risc0 proof demonstrating the need for deactivation.
- **Ownership Management:** Supports a secure two-step ownership transfer process, as well as the option for the owner to renounce control entirely.

Code Assessment. The RISC Zero Solana Verifier developers provided the source code of the solana verifier programs for the code review. The source code appears to be mostly original code written by the developers. It contains some documentation in the form of READMEs and documentation comments on functions and storage variables.

The source code contained a test suite, which the Veridise security analysts noted provided thorough coverage of functional workflows, as well as negative paths.

* The report is available here:
https://veridise.com/audits-archive/company/risc-zero/risc0-solana-2024-09-26/

Summary of Issues Detected. The security assessment uncovered 7 issues, 3 of which are assessed to be of low severity. Specifically, V-RSV-VUL-001 explains how verifiers disabled by the Router's emergency stop can still be invoked directly, and V-RSV-VUL-002 details how anyone can front run the initialization to seize Router ownership.

The Veridise analysts also identified 4 warnings, including V-RSV-VUL-004 where the emergency stop can be re-invoked on an already stopped verifier causing duplicate event emissions, and V-RSV-VUL-006 that notes the presence of outdated comments from a previous implementation causing confusion and misinterpretation.

Boundless has fixed all of the identified issues.

Recommendations. After conducting the assessment of the protocol, the security analysts had the following suggestion to improve the RISC Zero Solana Verifier.

Clarify integration aspects. The Router permits the contract owner to disable specific verifiers if they are believed to be malfunctioning. While disabling a malicious verifier is beneficial, protocols that integrate with the Router may fail to function correctly if this scenario is not accounted for. It is therefore recommended to explicitly document and highlight the potential issues that may arise during integration with the Router.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
RISC Zero Solana Verifier	`3be7250`	Rust	Solana

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Oct. 6–Oct. 7, 2025	Manual & Tools	2	4 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	0	0	0
Medium-Severity Issues	0	0	0
Low-Severity Issues	3	3	3
Warning-Severity Issues	4	4	4
Informational-Severity Issues	0	0	0
TOTAL	7	7	7

Table 2.4: Category Breakdown.

Name	Number
Maintainability	3
Access Control	2
Usability Issue	1
Data Validation	1

3 Security Assessment Goals and Scope

3.1 Security Assessment Goals

The engagement was scoped to provide a security assessment of RISC Zero Solana Verifier's source code. During the assessment, the security analysts aimed to answer questions such as:

- Are all ownership features in the `ownable` library implemented correctly and securely?
- Is access control properly enforced across all instructions that modify program state, or perform privileged actions?
- Do the programs perform the necessary validations and enforce appropriate constraints on the accounts involved in each instruction?
- Can verifiers with malicious or unauthorized logic be added to the verifier router?
- Does the verifier router correctly route verification requests based on the provided selector?
- Is the emergency stop event emitted correctly? And does it contain all the necessary contextual information to support monitoring and debugging?
- Does the system properly prevent access to verifiers that have been emergency-stopped, both through the router and via direct calls?
- Do the programs exhibit any known vulnerability patterns commonly associated with Solana or Anchor programs?
- Are the programs vulnerable to any common smart contract vulnerabilities, such as front-running or arithmetic underflows?

3.2 Security Assessment Methodology & Scope

Security Assessment Methodology. To address the questions above, the security assessment involved a comprehensive manual review of the Solana program source code.

Scope. The primary scope of this security assessment encompasses the Solana program source files implementing the `ownable` library and the `verifier_router` components of the RISC Zero Solana Verifier.

- `solana-ownable/ownable/src/lib.rs`
- `solana-ownable/ownable-macro/src/lib.rs`
- `solana-verifier/programs/verifier_router/src/client.rs`
- `solana-verifier/programs/verifier_router/src/lib.rs`
- `solana-verifier/programs/verifier_router/src/estop/events.rs`
- `solana-verifier/programs/verifier_router/src/estop/mod.rs`
- `solana-verifier/programs/verifier_router/src/router/error.rs`
- `solana-verifier/programs/verifier_router/src/router/mod.rs`
- `solana-verifier/programs/verifier_router/src/state/mod.rs`

Additionally, the secondary scope of this security assessment involved verifying that no functional changes had been introduced to the `groth16_verifier` implementation since the previous audit, which concluded at commit `6db7460`. The related files are listed below.

- `solana-verifier/programs/groth_16_verifier/src/client.rs`
- `solana-verifier/programs/groth_16_verifier/src/error.rs`
- `solana-verifier/programs/groth_16_verifier/src/lib.rs`
- `solana-verifier/programs/groth_16_verifier/src/v3_test_receipt.rs`
- `solana-verifier/programs/groth_16_verifier/src/vk.rs`

Methodology. Veridise security analysts reviewed the reports of previous audits for RISC Zero Solana Verifier, inspected the provided tests, and read the RISC Zero Solana Verifier documentation. They then began an in-depth manual review of the code.

During the security assessment, the Veridise security analysts met with the RISC Zero Solana Verifier developers to ask questions about the code, and gain an understanding of the program logic, intended assumptions, and use cases.

3.3 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

The likelihood of a vulnerability is evaluated according to the Table 3.2.

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

The impact of a vulnerability is evaluated according to the Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Security Review Assumptions

4.1 Operational Assumptions

In addition to assuming that any out-of-scope components behave correctly, Veridise analysts assumed the following properties held when modeling security for RISC Zero Solana Verifier.

- The **Verifier Router Owner** is expected to act honestly and fulfill their responsibilities, including refraining from intentionally adding bogus verifiers or transferring ownership to an untrusted party. They must also avoid halting the Router for specific selectors without a compelling justification.
- The **Verifier Router Upgrade Authority** — the account authorized to upgrade the main Router contract — is expected to act honestly and responsibly, ensuring that upgrades do not introduce vulnerabilities, whether intentionally or inadvertently.

Exploits that depend on the above roles acting beyond their intended privileges are considered out of scope.

Operational Recommendations. Highly-privileged operations should be operated by a multi-sig contract or decentralized governance system. These operations should be guarded by a timelock to ensure there is enough time for incident response. Highly-privileged operations should be tested in example scenarios to ensure the role operators are available and ready to respond when necessary.

Full validation of operational security practices is beyond the scope of this review. Users of the protocol should ensure they are confident that the operators of privileged keys are following best practices such as:

- Never storing a protocol key in plaintext, on a regularly used phone, laptop, or device, or relying on a custom solution for key management.
- Using separate keys for each separate function.
- Storing multi-sig keys in a diverse set of key management software/hardware services and geographic locations.
- Enabling 2FA for key management accounts. SMS should *not* be used for 2FA, nor should any account which uses SMS for 2FA. Authentication apps or hardware are preferred.
- Validating that no party has control over multiple multi-sig keys.
- Performing regularly scheduled key rotations for high-frequency operations.
- Securely storing physical, non-digital backups for critical keys.
- Actively monitoring for unexpected invocation of critical operations and/or deployed attack contracts.
- Regularly drilling responses to situations requiring emergency response such as pausing/unpausing.

5 Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 5.1 summarizes the issues discovered:

Table 5.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-RSV-VUL-001	Disabled verifier still accessible via direct calls	Low	Fixed
V-RSV-VUL-002	Potential front-running of Router's . . .	Low	Fixed
V-RSV-VUL-003	Improper ownership renouncement . . .	Low	Fixed
V-RSV-VUL-004	Duplicate EmergencyStopEvent emissions . . .	Warning	Fixed
V-RSV-VUL-005	Maintainability concerns	Warning	Fixed
V-RSV-VUL-006	Mismatch between comments and actual . . .	Warning	Fixed
V-RSV-VUL-007	Seal encoding should reject oversize inputs	Warning	Fixed

5.1 Detailed Description of Issues

5.1.1 V-RSV-VUL-001: Disabled verifier still accessible via direct calls

Severity	Low	**Commit**	3be7250
Type	Access Control	**Status**	Fixed
Location(s)	`solana-verifier/programs/groth_16_verifier/[...]/lib.rs:84-85`		
Confirmed Fix At	`pull/18throughb9de2dd`, b9de2dd		

Description

The `verify()` function of the `groth16` verifier is publicly callable by any account, which allows end-users to directly invoke the verifier logic. The Router is intended to serve as a control layer, maintaining the list of valid verifiers and supporting emergency stops via `estopped`. However, because `verify()` is not restricted to calls originating from the Router, a verifier that has been disabled in the Router can still be invoked directly through the verifier program itself.

This design undermines the Router's security guarantees and allows circumvention of critical safety controls.

Impact

- **Bypassing E-Stop:** A verifier that has been disabled in the Router remains callable directly, nullifying the intended protection of the emergency stop mechanism.
- **Security model confusion:** Off-chain integrators and operators may assume that disabling a verifier in the Router renders it completely inaccessible, which is not the case.
- **Potential attack surface:** If a verifier is disabled due to compromise, attackers may continue exploiting it by calling `verify()` directly.

Recommendation

Restrict access to `verify()` so that it can only be invoked by the Router program. Possible mitigations include:

- **Access control check:** Add a check inside `verify()` that validates the caller is the Router account.
- **Explicit documentation:** If public access is intentionally allowed, clearly document that the Router's `estopped` mechanism does not fully prevent verification, so operators and integrators are not misled.

Restricting invocation to the Router ensures that emergency stop controls are enforceable and prevents disabled verifiers from being reused in unintended ways.

Developers Response

Documentation has been added on the groth16 verify function and the readme to ensure developers integrating understand the risks of not integrating via the router, and are also directed to use an existing singleton router deployment.

5.1.2 V-RSV-VUL-002: Potential front-running of Router's initialization

Severity	Low	**Commit**	3be7250
Type	Access Control	**Status**	Fixed
Location(s)	`solana-verifier/programs/verifier_router/[...]/mod.rs:50-51`		
Confirmed Fix At	`pull/18through9360877`, 9360877		

Description

The `initialize` instruction creates the Router account as a PDA using fixed seeds `[b"router"]` and sets the caller (`authority`) as the Router owner. Because the PDA derivation does not include any identifier tied to the intended owner, anyone can call `initialize()` and become the Router owner as long as the Router PDA does not yet exist. An attacker can therefore front-run legitimate deployment and claim the Router, register malicious verifiers, or otherwise abuse ownership.

Impact

- Complete takeover of Router before legitimate owner initializes it.
- Register malicious verifiers for selectors that the legitimate owner will expect to control, allowing spoofed or compromised verification.
- Bypass or negate emergency controls and trust assumptions relying on Router ownership.
- Downtime or loss of integrity for systems that depend on Router as the authoritative registry.

Recommendation

1. **Bind the PDA to the intended owner in the seeds:**
   Include the intended authority public key in the PDA seeds so the Router PDA can only be derived/initialized for that owner.
2. **Require an out-of-band known deployer/administrator key:**
   If a single privileged deployer should perform initialization, require that `authority` equals a pre-determined public key hard-coded or stored in an on-chain config account.

Developers Response

The developers addressed the issue by hard-coding an initial owner set via compile time environment variables. A constraint enforces the deployer (and initial owner) match this value.

5.1.3 V-RSV-VUL-003: Improper ownership renouncement results in DoS risk

Severity	Low	**Commit**	3be7250
Type	Maintainability	**Status**	Fixed
Location(s)	`solana-verifier/programs/verifier_router/[...]/lib.rs:110-111`		
Confirmed Fix At	`pull/18throughd35213f`, d35213f		

Description

The `renounce_ownership()` function in the Router contract introduces problematic behavior. If invoked, it permanently removes the Router's ability to add or disable verifiers, effectively freezing the contract's management functionality. This resembles the default OpenZeppelin Ownable pattern but appears to have been included without a clear consideration of its implications for this specific context.

While the Router contract can technically be recovered by upgrading it to a new version (since a separate upgrade authority exists), this requires redeployment and migration of integrations. Such a process is disruptive, particularly if external protocols already rely on the current Router instance. If the Router were to be made fully immutable in the future (e.g., by revoking upgrade authority), invoking `renounce_ownership()` would result in a permanent denial of service for all integrated protocols.

Additionally, the existence of the function is misleading: despite "renouncing ownership," the upgrade authority can still regain control, meaning the function does not guarantee true decentralization or immutability.

Impact

- **Denial of Service (DoS):** A single call to `renounce_ownership()` can freeze critical management functionality, preventing the addition or disabling of verifiers.
- **Ecosystem-wide Disruption:** Protocols integrated with the Router would be forced to migrate to a new Router instance if recovery via upgrade is required, causing significant coordination overhead.
- **Misleading Decentralization:** Users may assume that ownership has been fully renounced, while in practice the upgrade authority retains control. This can create trust and governance concerns.
- **Permanent Risk (if immutable):** If upgrade authority is revoked and the function has been called, the Router may be permanently unusable, leading to irrecoverable protocol breakage.

Recommendation

It is recommended to remove the `renounce_ownership()` method, as it does not align well with the intended use case of the Router.

Developers Response

The developers removed `renounce_ownership()` from the router, as recommended.

5.1.4 V-RSV-VUL-004: Duplicate EmergencyStopEvent emissions due to idempotent estop functions

Severity	Warning	**Commit**	3be7250
Type	Usability Issue	**Status**	Fixed
Location(s)	`solana-verifier/programs/verifier_router/src/[...]/mod.rs:95`		
Confirmed Fix At	`pull/18through66021b4`, 66021b4		

Description

The contract provides two functions for disabling verifiers:

- `emergency_stop_by_owner()` - allows the contract owner to permanently disable a verifier.
- `emergency_stop_with_proof()` - allows anyone to disable a verifier by providing a valid proof of exploit.

Both functions are idempotent: once a verifier is marked as `estopped = true`, subsequent calls to the same function on the same verifier have no additional effect on state. However, each call will still emit an `EmergencyStopEvent`.

As a result, the same verifier can generate multiple identical stop events, even though the underlying action has already been taken. This may cause confusion for operators or automated monitoring systems that rely on event logs to track contract activity.

Impact

On-chain state remains correct (a verifier cannot be re-enabled), but event logs become unreliable indicators of system health. During incident response, it may be harder to distinguish the initial disabling event from redundant emissions.

Recommendation

To mitigate this issue, consider adding a guard clause to both `emergency_stop_by_owner()` and `emergency_stop_with_proof()` to check if the verifier is already `estopped` before emitting an event. For example:

```
if verifier.estopped {
    return Ok(()); // No-op, avoid duplicate event
}
```

Developers Response

The developers have added the recommended guards to both functions in question.

5.1.5 V-RSV-VUL-005: Maintainability concerns

Severity	Warning	**Commit**	3be7250
Type	Maintainability	**Status**	Fixed
Location(s)	`solana-verifier/programs/` ▶ `groth_16_verifier/src/` • `client.rs:33-44` • `error.rs:22, 24` • `lib.rs:253-258` ▶ `verifier_router/src/` • `client.rs:57` • `router/` ∗ `error.rs:24, 26, 28, 34`		
Confirmed Fix At	`pull/18throughaf90140`, af90140		

Description In several parts of the codebase, Veridise analysts identified maintainability issues including outdated or incorrect comments, unoptimized logic, and unused code. These issues can create confusion for developers and make the codebase harder to maintain and extend. The affected locations are listed below.

- **Unused code**
  - `solana-verifier/programs/groth_16_verifier/src/error.rs`.
    - `G1CompressionError`, `G2CompressionError` are unused.
  - `solana-verifier/programs/verifier_router/src/router/error.rs`.
    - `VerifierInvalidLoader`, `SelectorInvalid`, `SelectorNotFound`, `Overflow` are unused.
  - `solana-verifier/programs/groth_16_verifier/src/client.rs`.
    - The struct `VerifyingKeyJson` is unused.
- **Unoptimized logic**
  - In `solana-verifier/programs/verifier_router/src/state/mod.rs`, on line 44 when allocating space for the router account during initialization, an extra 4 bytes of space is allotted to accommodate the verifier count. However, this field is no longer implemented in the current iteration, making the additional allocation unnecessary and safe to remove.
  - In `encode_seal_with_selector()`, on line 57, the `selector` is already of type `Selector` (same as `Seal::Selector`), but it is converted again with `try_into()`.
  - In `solana-verifier/programs/verifier_router/src/router/mod.rs`, in `AddVerifier`, the `router` account is marked as `mut` even though it is only read and never modified.
- **Inconsistent coding patterns**
  - In `solana-verifier/programs/verifier_router/src/router/mod.rs`, within `verify()` (lines 222 and 225), the `if (condition) { ... }` statements can be replaced with `require!(condition)`, to match the pattern used in `solana-ownable`.

- **Misleading names**
  - In `solana-verifier/programs/verifier_router/src/client.rs`, in `encode_seal()`, the input parameter name `seal` is confusing in the program's context. The resulting `Seal` struct contains a selector and a proof, but the `seal` parameter represents only the proof (with no selector). It should instead be named `proof` to avoid confusion.
  - In `solana-verifier/programs/groth_16_verifier/src/lib.rs`, `to_field_element()` does not call `verify_scalar_in_field()` on its output to ensure it is a valid field element. The name is therefore misleading. Although this validation occurs later in `verify_groth16()`, in isolation the function's behavior doesn't fully match its name. Consider renaming it to `to_field_element_unchecked()` to avoid incorrect assumptions.
- **Implicit assumptions**
  - The `negate_g1()` function assumes the y-coordinate is always within the base field range and does not validate it. While this is safe for its current use, adding a comment or an explicit range check would improve maintainability and reduce the risk of misuse if the function is reused elsewhere.

Impact

1. Unused constructs may become out of sync with the rest of the project, leading to errors if used in the future.
2. Inconsistent coding patterns, misleading names and implicit assumptions reduce clarity and create confusion during reviews and refactoring.
3. Unoptimized logic can unnecessarily increase transaction costs.

Recommendation Remove the unused constructs and address the other maintainability concerns.

Developer Response The developers have addressed the maintainability concerns by making the suggested changes.

5.1.6 V-RSV-VUL-006: Mismatch between comments and actual business logic

Severity	Warning	Commit	3be7250
Type	Maintainability	Status	Fixed
Location(s)	`solana-verifier/programs/verifier_router/src/` ▶ `estop/mod.rs:43`		
Confirmed Fix At	`pull/18through0ca34dd`, 0ca34dd		

Description In `solana-verifier/programs/verifier_router/src/state/mod.rs`, the comments on the `VerifierRouter` account type incorrectly state that the account maintains a verifier count along with ownership information. This is not accurate as the verifier count has been removed in a previous iteration. References to this field still appear in other parts of the codebase, suggesting that the related comments and documentation are not fully updated to reflect the change.

```
/// Main router account storing ownership and verifier count
///
/// This account maintains the registry of verifiers and implements ownership controls
/// for administrative operations.
///
///
/// # Fields
/// * `ownership` - Stores the current and pending owner information using the Ownable trait
#[account]
#[derive(Ownable)]
pub struct VerifierRouter {
    pub ownership: Ownership,
}
```

Similarly, in `solana-verifier/programs/verifier_router/src/estop/mod.rs`, the comment on line 43 mentions that the `verifier_entry` account will be closed when a verifier is successfully e-stopped and the lamports will be refunded to the caller. This is not true, since the `verifier_entry` account is **not** closed when a verifier is emergency stopped and instead the `estopped` field is set to mark its disabled status.

```
/// This entry will be closed and refunded to the caller on successful stop
#[account(
    mut,
    seeds = [
        b"verifier",
        selector.as_ref()
    ],
    bump,
    constraint = verifier_entry.selector == selector,
    constraint = verifier_entry.verifier == verifier_program.key(),
)]
pub verifier_entry: Account<'info, VerifierEntry>,
```

Impact Incorrect comments make the codebase harder to read and reason about, especially for new contributors. Over time, they can introduce subtle confusion and make refactoring or

development more error-prone due to incorrect assumptions.

Recommendation Update the comments to match current behavior. Remove all mentions of a `verifier_count` and clarify that `estop` sets `estopped = true` without any accounts being closed.

Developer Response The developers updated the comments, as recommended.

5.1.7 V-RSV-VUL-007: Seal encoding should reject oversize inputs

Severity	Warning	**Commit**	3be7250
Type	Data Validation	**Status**	Fixed
Location(s)	`solana-verifier/programs/verifier_router/[...]/client.rs:45`		
Confirmed Fix At	`pull/18throughed2449b`, ed2449b		

Description The `encode_seal_with_selector` function currently accepts any input longer than 256 bytes and silently truncates it to that length. Because a valid proof must always be exactly 256 bytes, this behavior can hide malformed or corrupted inputs.

Additionally, the function returns a generic `Err(())` for all failure cases, which makes it difficult to distinguish between a short input, a failed slice conversion, or an invalid selector.

Impact Allowing oversize inputs to pass through can cause distinct byte arrays to be interpreted identically after truncation, obscuring encoding or serialization errors and making debugging more difficult.

Recommendation Enforce a strict length check that requires the input to be exactly 256 bytes and reject anything else.

Consider replacing the generic error with specific error messages to clearly indicate the type of failure encountered.

Developer Response The developers now enforce a strict length check, as recommended.