# STRYNER Whitepaper v1.0

## Part I — Executive Summary

**Version:** 1.0

---

# Executive Summary

## Introduction

STRYNER is a Telegram-native Web3 fitness platform built around one core principle:

> **Discipline should have measurable value.**

Unlike traditional fitness applications that reward activity with points or badges alone, STRYNER introduces commitment-based challenges where users voluntarily participate, verify their physical activity, and earn rewards through transparent, rule-based competition.

The platform is designed around **fairness**, **privacy**, and **user ownership**.

No biometric profiling.

No GPS tracking.

No invasive analytics.

No unnecessary personal information.

Users remain in control of both their identity and their digital assets.

---

# Vision

The long-term vision of STRYNER is to become the leading infrastructure for commitment-based fitness challenges inside Telegram and the TON ecosystem.

Rather than creating another move-to-earn application, STRYNER aims to build an ecosystem where users voluntarily commit to personal goals and where blockchain technology acts only as the settlement layer—not as the product itself.

Blockchain should remain invisible for users who simply want to participate in challenges.

---

# Mission

STRYNER exists to answer one simple question:

> **What if discipline itself became a measurable digital asset?**

The platform encourages consistency instead of speculation.

Every feature is designed around three principles:

- Commitment
- Transparency
- Fair competition

Unlike traditional GameFi products, STRYNER avoids artificial inflation mechanics and focuses on sustainable challenge economics.

---

# Product Overview

STRYNER consists of several independent but connected components.

### Telegram Mini App

The Mini App serves as the primary user interface.

Users can:

- create an account
- join challenges
- monitor progress
- connect a TON wallet
- view rankings
- claim rewards
- invite referrals

No sensitive health information is processed inside Telegram.

---

### Android Companion App

The Companion App is responsible only for activity verification.

It synchronizes verified daily step counts from Android Health Connect.

The application intentionally does not function as a complete fitness tracker.

Its only responsibility is secure synchronization.

---

### Backend Infrastructure

Backend services perform:

- challenge management
- participant verification
- anti-cheat validation
- reward calculation
- synchronization
- statistics generation

The backend never receives raw Health Connect records.

Only verified daily totals are transmitted.

---

### Blockchain Layer

The TON blockchain is used exclusively as the settlement layer.

It is **not** responsible for gameplay.

Instead it provides:

- transparent reward settlement
- challenge escrow
- treasury accounting
- immutable payout history

---

# Core Principles

Every engineering decision follows the same priorities.

## Privacy First

Only the minimum amount of data required to operate the platform is collected.

The system intentionally avoids:

- GPS
- Heart Rate
- Calories
- Sleep
- Contacts
- Advertising IDs
- Device IDs
- Behavioral profiling

---

## Fair Competition

Challenge outcomes should depend exclusively on verified physical activity.

Artificial advantages, automation and manipulated activity records are rejected by design.

---

## Non-Custodial Philosophy

STRYNER uses a non-custodial escrow model.

Users authorize transactions directly from their own TON wallet.

The platform never obtains access to private keys.

Challenge funds remain controlled by smart contract logic rather than by STRYNER.

---

## Transparency

Economic rules are published before joining every challenge.

Users know:

- entry fee
- platform fee
- treasury allocation
- referral allocation
- payout rules

before participating.

---

# Current Development Status

Current implementation includes:

✅ Telegram Mini App

✅ Android Companion

✅ Health Connect synchronization

✅ Backend infrastructure

✅ Referral system

✅ XP progression

✅ Anti-cheat foundation

✅ Smart contract escrow (live on mainnet)

✅ Mainnet settlement (live)

In development:

- iOS Companion
- Premium Features
- Advanced Statistics

## Part II — Architecture

# 2.1 Architecture Decisions (ADR)

## Why Telegram First?

One of the earliest architectural decisions made during the STRYNER design process was to use Telegram as the primary application environment instead of building a traditional standalone mobile application.

This decision was not driven by development speed alone. It was primarily motivated by distribution, user acquisition costs and friction reduction.

Traditional fitness applications require users to:

- download an application,
- create a new account,
- verify an email address,
- remember another password,
- complete onboarding,
- only then join a challenge.

Each of these steps increases user drop-off.

Telegram eliminates nearly all of them.

A new user can open STRYNER directly inside an application they already use every day.

Authentication is handled through Telegram.

No email registration is required.

No password management is required.

No additional messaging platform is needed.

The Mini App therefore becomes part of the existing Telegram ecosystem rather than competing with it.

From a business perspective this significantly reduces Customer Acquisition Cost (CAC) while improving activation and retention.

---

## Why a Companion App?

Telegram Mini Apps cannot access Android Health Connect.

This limitation forced an architectural decision.

Three approaches were evaluated.

### Option A — Manual Step Entry

Rejected.

Manual input introduces obvious fraud opportunities and completely undermines competitive integrity.

---

### Option B — Google Fit Synchronization

Rejected.

Google Fit has entered maintenance mode and no longer represents Google's strategic health platform.

Its long-term future is uncertain.

Building new infrastructure on a deprecated ecosystem would introduce unnecessary migration costs.

---

### Option C — Android Companion Application

Selected.

The Companion App performs exactly one responsibility:

Secure verification and synchronization of daily step counts.

This separation produces several important benefits.

First, it allows the Telegram Mini App to remain lightweight and platform-independent.

Second, Health Connect permissions are isolated inside a dedicated application.

Third, future support for Apple Health can be implemented without redesigning the Telegram application.

The Companion App therefore acts as a trusted bridge between operating-system health services and the STRYNER backend.

---

# Why Health Connect?

Health Connect was selected because it is now Google's recommended platform for health data interoperability.

Unlike Google Fit APIs, Health Connect provides:

- standardized permission management,
- local data ownership,
- stronger privacy guarantees,
- vendor-neutral integration,
- long-term platform support.

Most importantly, STRYNER intentionally requests access to only a single record type:

**Steps.**

No additional health metrics are required for challenge verification.

This greatly reduces regulatory exposure while simplifying user consent.

---

# Why Not Build Everything On-Chain?

A common misconception in Web3 is that every component should be decentralized.

STRYNER deliberately rejects this approach.

The blockchain is used only where decentralization provides measurable value.

Health verification, anti-cheat analysis and synchronization remain off-chain because they require high-frequency processing, flexible updates and interaction with external operating systems.

Attempting to execute these processes inside smart contracts would significantly increase transaction costs while providing little additional trust.

Instead, blockchain is reserved for:

- escrow,
- settlement,
- transparency,
- immutable payout history.

This hybrid architecture combines the operational efficiency of centralized services with the trust guarantees of decentralized settlement.

---

# Why Non-Custodial Escrow?

Originally, STRYNER considered a custodial settlement wallet controlled by the backend.

Although technically simpler, this approach introduced several disadvantages:

- operator custody,
- higher legal exposure,
- increased attack surface,
- greater user trust requirements.

Following architectural review, the project adopted a non-custodial direction.

In the production model, users authorize challenge participation directly from their own TON wallets through TON Connect.

Funds are transferred into a dedicated smart contract escrow.

The smart contract enforces challenge rules and distributes rewards according to predefined settlement logic.

At no point does STRYNER obtain access to participants' private keys or maintain the ability to move user funds unilaterally.

This design substantially improves transparency while reducing custodial risk and aligning the platform with Web3 self-custody principles.

# 2.2 Design Goals

## Purpose of This Chapter

Before describing the technical architecture of STRYNER, it is important to define the objectives that guided its design.

Every architectural decision represents a trade-off. Choosing one solution often means accepting limitations in another area. Rather than optimizing for every possible use case, STRYNER was designed around a small number of clearly defined priorities that influence every layer of the platform—from user onboarding and backend services to blockchain settlement and privacy.

These goals serve as the foundation for evaluating future architectural changes. Any new feature or modification should support these objectives rather than compromise them.

---

# Primary Design Goals

## 1. Fair Competition

The primary objective of STRYNER is to create a competitive environment in which challenge outcomes are determined exclusively by verified physical activity.

The platform is intentionally designed to minimize opportunities for manipulation through manual data entry, automation, unauthorized software or unfair advantages. Competitive integrity takes precedence over convenience. Whenever a trade-off exists between usability and fairness, the architecture favors fairness.

This principle influences anti-cheat mechanisms, challenge verification, synchronization logic and future smart contract settlement.

---

## 2. User Privacy

Privacy is treated as a fundamental architectural requirement rather than an optional feature.

STRYNER follows the principle of collecting only the minimum amount of information necessary to operate the platform. The system intentionally avoids access to sensitive health metrics, precise location data and behavioral profiling.

By limiting data collection at the architectural level, the platform reduces regulatory complexity, minimizes the impact of potential data breaches and gives users greater confidence in how their information is handled.

Privacy is therefore achieved not only through policy but through technical design.

---

# 3. User Ownership

Users should retain ownership of both their identity and their digital assets whenever technically possible.

Authentication is separated from financial authorization, and blockchain interactions are designed around a non-custodial model. Participants authorize transactions directly through their own wallets instead of transferring asset control to the platform.

This approach aligns STRYNER with the broader principles of self-custody within the Web3 ecosystem and reduces the level of trust required between users and the platform operator.

---

# 4. Low User Friction

A platform designed to encourage long-term consistency must minimize unnecessary barriers to participation.

The onboarding process should require as few steps as possible while maintaining appropriate security standards. Users should be able to discover challenges, register, connect a wallet and begin participating without navigating complex registration flows or managing additional credentials.

Telegram was selected as the primary interface because it significantly reduces friction compared to traditional standalone applications.

Lower onboarding friction directly improves activation rates, user retention and overall accessibility.

---

# 5. Modular Architecture

STRYNER is intentionally divided into independent components with clearly defined responsibilities.

The Telegram Mini App, Android Companion, backend services and blockchain layer each perform specialized tasks and communicate through controlled interfaces.

This modular approach provides several long-term advantages:

- independent deployment of services,

- easier testing and maintenance,
- simplified security auditing,
- isolated failure domains,
- faster feature development.

No individual component should become responsible for multiple unrelated concerns.

---

# 6. Scalable Infrastructure

The platform is designed to support gradual growth without requiring fundamental architectural redesign.

Infrastructure components should scale independently based on demand. Backend services, databases, static hosting and blockchain settlement are intentionally separated to prevent bottlenecks and reduce operational complexity.

Scalability is considered not only in terms of performance but also operational cost. The architecture aims to maintain predictable infrastructure expenses as the user base grows.

---

# 7. Transparent Economics

Economic rules must be understandable before a participant joins any challenge.

Users should always be able to determine:

- participation requirements,
- entry fee,
- platform fee,
- treasury allocation,
- referral allocation,
- reward calculation,
- settlement conditions.

No hidden mechanisms or discretionary operator decisions should influence financial outcomes after a challenge has started.

Transparency strengthens user trust and simplifies future auditing of platform economics.

---

# 8. Long-Term Maintainability

Software quality is measured not only by how quickly new features are delivered but also by how safely the system can evolve.

The architecture therefore prioritizes readability, modularity and predictable behavior over unnecessary technical complexity.

Whenever multiple implementation approaches are available, STRYNER favors solutions that reduce future maintenance effort, simplify debugging and improve long-term reliability.

This principle applies equally to backend services, database design, mobile applications and smart contract architecture.

---

## Design Goal Summary

Every future feature introduced into STRYNER should support at least one of the following objectives without compromising the others:

- Fair Competition
- User Privacy
- User Ownership
- Low User Friction
- Modular Architecture
- Scalable Infrastructure
- Transparent Economics
- Long-Term Maintainability

These design goals serve as the architectural compass for the continued evolution of the platform. Features that conflict with these principles should be carefully evaluated before implementation and justified through a formal Architecture Decision Record (ADR).

# 2.3 Non-Goals

## Purpose of This Chapter

A well-defined architecture is determined not only by what a system is designed to accomplish, but also by what it intentionally excludes.

Clearly defining non-goals prevents uncontrolled feature expansion, reduces architectural complexity and establishes realistic expectations for users, partners and future contributors.

Every feature that falls outside the scope defined below should be considered a separate product decision rather than a natural extension of the current platform.

---

# Product Scope

STRYNER is a commitment-based challenge platform.

Its primary responsibility is to facilitate fair participation in physical activity challenges, verify predefined activity metrics and settle challenge outcomes through transparent rules.

The platform is intentionally limited to this domain.

Although additional functionality may be introduced in future releases, the current architecture deliberately avoids expanding into unrelated areas that would increase complexity without directly supporting the platform's core objectives.

---

# STRYNER Is Not a Fitness Tracker

The platform is not intended to replace dedicated health or fitness applications.

Users should continue using specialized products for activity analysis, workout planning or health monitoring.

STRYNER does not aim to compete with applications such as Google Fit, Samsung Health, Apple Health or Garmin Connect.

Instead, it consumes only the minimum information required to verify challenge participation.

The platform therefore acts as a verification layer rather than a comprehensive fitness ecosystem.

---

# STRYNER Is Not a Social Network

Although participants may compete against one another, STRYNER is not designed as a social media platform.

The current architecture intentionally excludes features such as:

- public user feeds,
- private messaging,
- photo sharing,
- comments,
- content creation,
- follower systems,
- advertising timelines.

Telegram already provides mature communication capabilities, making duplication unnecessary.

This architectural decision reduces development complexity while allowing STRYNER to remain focused on challenge execution.

---

# STRYNER Is Not a Cryptocurrency Exchange

The platform is not intended to facilitate cryptocurrency trading, asset speculation or investment services.

Users cannot:

- exchange digital assets,
- perform market trading,
- create liquidity pools,
- speculate on token prices,
- access leveraged financial products.

Any blockchain interaction performed by STRYNER exists solely to support challenge settlement and reward distribution.

---

# STRYNER Does Not Custody User Assets

One of the core architectural boundaries is the absence of custodial wallet management.

The production architecture is designed so that participants authorize blockchain transactions directly through their own wallets.

Private keys are never transferred to STRYNER infrastructure.

The platform therefore does not function as a wallet provider, asset custodian or financial intermediary.

---

# STRYNER Does Not Monetize Personal Data

The business model of STRYNER is intentionally separated from personal data collection.

The platform does not sell user information to advertisers, data brokers or third-party analytics providers.

No advertising profiles are generated from health activity.

No behavioral scoring is created for marketing purposes.

User information exists exclusively to support challenge functionality and platform operations.

---

# STRYNER Does Not Track User Location

Geographic location is not required to determine challenge completion.

Accordingly, the platform intentionally avoids collecting:

- GPS coordinates,

- travel history,
- route recordings,
- location timelines,
- geofencing events,
- nearby device information.

Removing location tracking significantly improves privacy while reducing regulatory obligations associated with continuous location monitoring.

---

# STRYNER Does Not Perform Medical Analysis

The platform should never be interpreted as a medical application.

It does not diagnose, monitor or evaluate medical conditions.

The platform does not calculate:

- cardiovascular health,
- calorie expenditure,
- body composition,
- sleep quality,
- recovery metrics,
- training effectiveness.

Challenge verification should not be interpreted as evidence of health status or medical fitness.

---

# STRYNER Is Not an Identity Provider

User authentication relies on external trusted providers.

Telegram authenticates the user within the Mini App.

TON Connect authenticates wallet ownership.

Health Connect provides verified activity records.

STRYNER coordinates these systems but does not attempt to replace them with proprietary identity infrastructure.

---

# STRYNER Is Not an Anti-Cheat Guarantee

Although considerable effort is invested in fraud detection, no software system can guarantee complete prevention of abuse.

Instead, STRYNER is designed to minimize incentives, reduce attack surfaces and continuously improve detection capabilities.

Anti-cheat mechanisms should therefore be understood as an evolving security process rather than an absolute guarantee.

Future platform updates may introduce additional verification techniques as new attack vectors emerge.

---

# Product Focus

The long-term success of STRYNER depends on maintaining a disciplined product scope.

Every new feature should strengthen one of the platform's primary objectives:

- fair competition,
- transparent settlement,
- user ownership,
- privacy,
- long-term consistency.

Features that do not directly support these objectives should be evaluated as independent products rather than incorporated into the core platform.

This disciplined approach reduces technical debt, improves maintainability and helps preserve the simplicity that defines the STRYNER architecture.

---

# Conclusion

Defining non-goals is not a limitation of the platform; it is a deliberate engineering strategy.

By explicitly identifying what STRYNER is **not**, the project protects itself from uncontrolled scope expansion, unnecessary complexity and conflicting product expectations.

These boundaries provide a stable foundation for future development while ensuring that every architectural decision remains aligned with the platform's long-term vision.

# 2.4 Core Engineering Principles

## Purpose of This Chapter

The long-term quality of a software platform is determined less by the technologies it uses and more by the engineering principles that guide every architectural decision.

Technologies evolve rapidly. Frameworks, programming languages and infrastructure providers change over time. Engineering principles, however, should remain stable throughout the lifetime of the platform.

The following principles define the standards that govern the design, implementation and future evolution of STRYNER. Every new feature, service or architectural modification should be evaluated against these principles before implementation.

---

# Principle 1 — Privacy by Design

Privacy is treated as a system property rather than a feature added after development.

Every component of STRYNER is designed to minimize the collection, processing and storage of personal information. Instead of relying solely on privacy policies or user agreements, the architecture itself restricts access to unnecessary data.

Where multiple implementation options exist, preference is given to the solution that requires the least amount of personal information.

Examples include limiting Health Connect access to step counts only, avoiding GPS tracking and excluding unnecessary health metrics from synchronization.

Privacy should be enforced through architecture before it is enforced through policy.

---

# Principle 2 — Least Privilege

Every component should receive only the permissions required to perform its specific responsibility.

No application, service or backend process should possess broader access than necessary.

Examples include:

- the Companion App accesses only Health Connect step records,
- the Telegram Mini App has no access to health data,
- backend services cannot access user wallet private keys,
- smart contracts cannot access personal user information.

Reducing privileges limits the potential impact of software defects, configuration errors and malicious attacks.

---

# Principle 3 — Separation of Responsibilities

Every component should perform one clearly defined function.

The platform intentionally separates:

- user interaction,
- activity verification,
- business logic,
- blockchain settlement,
- data persistence,
- infrastructure management.

This separation improves maintainability, simplifies testing and allows individual services to evolve independently without introducing unnecessary dependencies.

Whenever a component begins performing unrelated responsibilities, architectural refactoring should be considered.

---

# Principle 4 — Security by Default

The secure behavior of the platform should represent the default configuration rather than an optional setting.

Security mechanisms should require explicit justification before being disabled.

Examples include:

- encrypted HTTPS communication,
- authenticated API requests,
- wallet signature verification,
- permission-limited applications,
- server-side validation,
- immutable blockchain settlement.

Whenever security and convenience conflict, the preferred implementation should prioritize security unless a clear business justification exists.

---

# Principle 5 — Fail Secure

Unexpected failures should never create conditions that compromise fairness or security.

If verification cannot be completed, challenge progress should remain unchanged until verification succeeds.

If settlement cannot be completed, funds should remain protected rather than being distributed incorrectly.

If external dependencies become temporarily unavailable, the platform should prefer delayed processing over potentially incorrect processing.

System failures should degrade functionality, not trust.

---

# Principle 6 — Deterministic Business Logic

Challenge outcomes should always be reproducible from the same input data.

Given identical participant data and challenge rules, the platform should always calculate identical results regardless of execution time or infrastructure location.

This deterministic behavior simplifies auditing, debugging and future smart contract verification.

Business rules should never depend on hidden variables, manual intervention or subjective operator decisions.

---

# Principle 7 — Transparency

Users should understand how platform decisions are made.

Wherever possible, challenge rules, settlement logic and economic mechanisms should be visible before participation.

Transparency reduces disputes, increases confidence and simplifies external auditing.

Whenever automated decisions affect challenge outcomes, those decisions should be explainable using objective platform rules.

---

# Principle 8 — Modularity

The architecture should support independent evolution of its components.

Replacing one subsystem should require minimal changes to unrelated services.

Examples include:

- replacing infrastructure providers,
- introducing an iOS Companion App,
- updating smart contract implementations,
- expanding challenge verification methods.

A modular architecture reduces migration costs and enables continuous improvement without large-scale redesign.

---

# Principle 9 — Scalability Through Simplicity

Scalability should primarily result from architectural simplicity rather than operational complexity.

The platform should avoid unnecessary microservices, excessive abstraction or premature optimization.

Infrastructure should remain understandable by a small engineering team while supporting gradual user growth.

Whenever a simpler solution provides equivalent functionality, it should be preferred over a more complex alternative.

---

# Principle 10 — Technology Independence

Business rules should remain independent of individual vendors, frameworks or cloud providers.

Where practical, platform logic should be portable between infrastructure providers.

The choice of Render, Supabase, Cloudflare or any other service should represent an implementation decision rather than a dependency embedded into the business model.

This principle reduces vendor lock-in and simplifies future infrastructure migrations.

---

# Principle 11 — Evolution Without Disruption

The platform should be capable of evolving without forcing unnecessary changes upon existing users.

Whenever possible:

- APIs should remain backward compatible,
- database migrations should preserve existing data,
- new features should extend rather than replace existing functionality,
- infrastructure upgrades should avoid service interruptions.

Incremental improvement is preferred over disruptive redesign.

---

# Engineering Principles Summary

Every architectural decision inside STRYNER should satisfy the following principles:

- Privacy by Design
- Least Privilege
- Separation of Responsibilities
- Security by Default
- Fail Secure
- Deterministic Business Logic
- Transparency
- Modularity
- Scalability Through Simplicity
- Technology Independence
- Evolution Without Disruption

These principles form the engineering foundation of the platform. They are intended to remain stable over time, regardless of future technological changes, ensuring that STRYNER continues to evolve without compromising its core values or architectural integrity.

# 2.5 System Architecture Overview

*This section describes the architecture of the STRYNER platform at a conceptual level. Implementation details that are security-relevant are intentionally omitted.*

---

## 1. Design principles

STRYNER is built around four constraints that shaped every architectural decision:

**Verification over trust.** A step count that cannot be verified is worthless in a commitment system. Steps are read from the device's own health platform and filtered before they ever reach our servers.

**Data minimisation.** The platform collects the smallest amount of data that makes the product work. What is not collected cannot be leaked.

**Separation of concerns.** Movement tracking, application logic, and value settlement are handled by separate components with distinct trust requirements.

**No unnecessary custody.** Value settlement is on-chain, so the operator holds no keys to participant funds.

---

## 2. Security philosophy

STRYNER is designed on the assumption that any individual component may eventually fail, be compromised, or be modified by a determined user.

Security therefore does not rest on trusting any single system. It rests on independent verification layers, each of which would have to fail simultaneously for an incorrect outcome to occur.

The architecture reaches a state in which no single component — including the platform operator — can:

- alter a recorded challenge outcome,
- modify verified activity data,
- move participant funds,
- bypass settlement rules.

The final item is why settlement runs on smart contracts, deployed and live on mainnet.

---

# 3. System layers

STRYNER consists of four layers:

| | |
|---|---|
| Client | Telegram Mini App |
| Verification | Companion App (Android) |
| Services | Application backend |
| Settlement | TON blockchain |

Each layer has a single responsibility and communicates with its neighbours over a narrow interface.

---

# 4. Trust boundaries

Components operate under different trust assumptions, and the architecture treats them accordingly.

| Component | Trust level | Rationale |
|---|---|---|
| Telegram | Trusted identity provider | Authenticates users; the platform stores no credentials |
| Device operating system | Trusted source | Provides step data together with its recording provenance |
| Companion App | **Untrusted client** | Runs on user-controlled hardware and can be modified |
| Network | **Untrusted** | All traffic encrypted; contents assumed observable |
| Backend | Trusted service | Enforces challenge rules; performs independent validation |
| TON blockchain | Public, trust-minimised | Consensus assumed secure; state publicly verifiable |
| Settlement contract | Deterministic | Rules fixed at deployment and publicly auditable |
| User wallet | User controlled | Only the user can authorise transactions |

The classification of the Companion App is deliberate. Client-side filtering improves data quality and protects user privacy by discarding unverifiable records before transmission, but it is not a security control: software running on a user's device can always be modified. Every value the backend receives is therefore validated again server-side, without reference to what the client claims about itself.

---

# 5. Data flow

```
User walks
    ↓
```

```
  Device motion sensor
       ↓
  System health platform          ← recording method and source recorded here
       ↓
  Companion App                   ← reads, filters, discards unverifiable
records
       ↓
  Backend                         ← independent validation, anomaly detection
       ↓
  Challenge progress
       ↓
  Settlement
       ↓
  Participant claim
```

**Sensor to health platform.** The operating system records steps together with metadata describing how they were recorded and which application wrote them. The platform does not control this stage and does not need to.

**Health platform to Companion App.** The app reads step records and their provenance. Records that were manually entered, or written by unrecognised applications, are discarded on the device.

**Companion App to backend.** Only a verified daily total is transmitted, together with a user identifier and date. Source names, device details, and rejected records are not sent.

**Backend to challenge progress.** The backend validates the received value independently and applies challenge rules to determine daily qualification.

**Challenge to settlement.** At the end of a challenge, and after a fixed verification window, results are evaluated and allocations determined.

**Settlement to claim.** Participants withdraw what they are owed independently.

---

# 6. Verification layer — Companion App

Step verification is the foundation of the product, and it is deliberately separated from the application itself.

The Companion App is a native Android application, published on Google Play as a distinct application, that reads step data from the device's system health platform and synchronises it in the background.

## What it reads

**Step counts only.** The application does not access distance, calories, heart rate, location, exercise sessions, or any other health metric. This restriction is enforced at the permission level, not merely by convention.

## How steps are filtered

Filtering happens on the device, before any data is transmitted:

- **Recording method.** Only steps recorded automatically by the device or actively recorded by a fitness application are accepted. Manually entered values are discarded.

- **Source provenance.** Steps must originate from the system health platform or a verified fitness provider. Data written by unrecognised applications is discarded.
- **Deduplication.** When multiple applications write the same physical steps, only the primary system source is counted, so a step is never counted twice.

### What is transmitted

After filtering, the Companion App sends a minimal payload: a user identifier, a date, a verified step total, and a synchronisation source marker. It does not transmit device identifiers, source application names, raw pre-filter counts, or operating system details.

---

# 7. Services layer — backend

The backend handles challenge lifecycle, progress evaluation, experience points and ranks, referral tracking, and settlement preparation. It performs validation independent of the client.

It runs on managed infrastructure with a PostgreSQL database. The platform operates no analytics tooling of any kind: no tracking pixels, no behavioural analytics, no third-party telemetry.

System logs contain internal identifiers only. IP addresses, user agents, device identifiers, usernames, and full wallet addresses are not written to logs.

---

# 8. Settlement layer — TON blockchain

Challenge entries and settlements are denominated in GRAM.

### Current status

Settlement runs on-chain through per-challenge smart contracts, live on mainnet. The full lifecycle — join, settlement, claim, and refund/expiry — has been executed with real value on the live network.

The contracts have not yet undergone an independent security review; that review is outstanding and is treated as a residual risk in 3.9.13 and Part III.

### Settlement architecture

Settlement uses a per-challenge smart contract model:

- **One contract per challenge.** A factory deploys an independent contract for each challenge. A fault in one challenge cannot affect any other challenge or any other participant's funds.
- **Time-based state transitions.** Registration closes and settlement windows open based on blockchain time, not on instructions from the platform. The platform cannot shorten a verification window or settle a challenge early.
- **Verification window.** Settlement cannot occur until a fixed period after a challenge ends, long enough for every participant's local day to have closed in every time zone worldwide.
- **Pull-based claims.** The contract records what each participant is owed; participants claim independently. There is no distribution queue that can stall part-way through.

- **Guaranteed refunds.** If settlement is not delivered within the permitted window, the contract enters a refund state and every participant can recover their full entry. This path requires no action from the platform and cannot be blocked by it.
- **No operator withdrawal.** The contract has no function permitting the operator to withdraw participant funds. Once deployed, its rules cannot be altered.

Under this model the platform submits settlement results but never holds participant funds.

---

# 9. Challenge lifecycle

```
Registration      participants join, entries are locked
      ↓
Active period     daily step goals, verified on-device
      ↓
Verification      fixed window; all time zones close
      ↓
Settlement        results submitted, allocations recorded
      ↓
Claims            participants withdraw independently
```

If settlement is not delivered in time, the flow diverts to a refund state in which every participant recovers their entry in full.

---

# 10. Settlement model

Entries form a challenge pool. Participants recover a share of their own entry proportional to the days they completed; the unearned remainder forms the distributable pool.

The pool is allocated across four fixed envelopes covering qualified finishers, treasury, referrals, and platform operations. Envelope proportions are published and enforced by contract.

Two properties are deliberate:

**Partial credit.** A participant who completes most of a challenge recovers most of their entry. Failure is proportional, not total.

**No operator gain from failure.** Amounts that cannot be allocated — an unclaimed referral share, a challenge nobody completes — are redirected to participants, never to the operator. The platform's revenue does not increase when users fall short.

---

# 11. Failure behaviour

The system consistently prefers delayed processing over potentially incorrect processing.

| Condition | Behaviour |
|---|---|
| Health platform temporarily unavailable | Synchronisation is postponed. Step counts are never estimated or interpolated. |
| Companion App offline or closed | Step data remains in the device's health platform and is read on the next successful synchronisation. Nothing is lost. |
| Backend unreachable | The app retries. Verified totals are re-read from the health platform |

| Condition | Behaviour |
|---|---|
| | rather than cached and replayed. |
| Backend validation rejects a value | Progress for that day remains unrecorded until a valid value is received. |
| Blockchain temporarily unavailable | Settlement remains pending. Recorded challenge results are unaffected. |
| Settlement not delivered within its window | The contract enters its refund state and participants recover entries in full. |

No failure path results in an estimated step count, a partially completed distribution, or funds that cannot be recovered.

---

# 12. Operating costs

Two separate charges apply, and they are not interchangeable:

**Platform fee.** A percentage of the entry, covering backend infrastructure, verification services, development, and security review. This is platform revenue.

**Contract reserve.** A per-participant amount funding the specific challenge contract: blockchain storage, claim execution, refund execution, and closure. This is working capital for that contract, not platform revenue. Any surplus remaining after settlement returns to participants.

Both are displayed before a participant joins.

---

# 13. Privacy posture

- Step counts only; no other health data is accessed
- Filtering occurs on-device; unverifiable data is never transmitted
- No analytics or behavioural tracking of any kind
- No IP addresses, device identifiers, or user agents in logs
- Wallet addresses are masked in operational logs
- Account deletion available on request, with a published scope

---

# 14. External dependencies

The platform currently depends on the following external services:

| Dependency | Role |
|---|---|
| Telegram | Identity and application delivery |
| Google Play | Companion App distribution |
| Android health platform | Step data source |
| TON | Settlement network |
| Managed hosting and database providers | Backend and application infrastructure |

Dependencies are isolated behind internal interfaces so that replacing any one provider affects a bounded part of the system. Two dependencies are structural rather than substitutable: Telegram,

which defines the product's distribution model, and the device health platform, which is the only source of verifiable step provenance.

---

## 15. Security assumptions

The architecture assumes:

- Telegram authentication remains trustworthy for identity purposes
- The device health platform correctly isolates health permissions and records accurate provenance
- TON consensus remains secure
- Users are responsible for protecting their own wallet keys
- Client devices may be compromised or modified
- Network traffic should always be treated as hostile
- Any single server-side component may fail or be breached

Where an assumption cannot be guaranteed, the design compensates with an independent check rather than relying on the assumption holding.

---

## 16. Scalability

Current capacity has not been load-tested at scale, and the architecture is documented in terms of design assumptions rather than measured throughput.

The per-challenge contract model bounds on-chain cost per challenge rather than across the platform, so settlement scales with challenge count rather than total user base. Backend workload is dominated by daily synchronisation, which is evenly distributed across time zones rather than concentrated at peaks.

Growth beyond current deployment is expected to require horizontal scaling of application services, database indexing work, and distributed background processing. These are anticipated rather than implemented.

---

## 17. Development status

| Component | Status |
|---|---|
| Telegram Mini App | Live |
| Companion App | Published on Google Play |
| On-device step filtering | Live |
| Backend and challenge logic | Live |
| Free-to-play challenges | Live |
| Smart-contract settlement | Live |
| Paid challenges | Live (independent audit pending) |

---

## 18. Direction

Beyond the settlement migration, development is directed toward:

- iOS support for step verification
- Additional verified data providers
- Broader challenge formats, including team-based participation
- Public verifiability of settlement records

These are directional, not commitments. Timelines depend on the completion and review of the settlement contracts, which take precedence over feature work.

# Part III — Security Architecture

## 3.1 Security Objectives

---

# Purpose of This Chapter

The purpose of the STRYNER security architecture is not to eliminate all possible risk. No software system can honestly make such a claim.

Instead, the objective is to identify realistic threats, reduce their probability, limit their impact, and ensure that no single failure can compromise the fairness, integrity, or financial security of the platform.

Security is therefore treated as a continuous engineering discipline rather than a fixed feature that can be completed.

Every architectural decision described throughout this document should ultimately contribute to one or more of the security objectives defined below.

---

# Security as a Product Requirement

Within STRYNER, security is considered a functional requirement rather than an operational afterthought.

Challenge fairness, participant trust and financial settlement all depend on the platform behaving predictably even when individual components fail, external services become unavailable, or users attempt to manipulate the system.

For this reason, security considerations influence every architectural layer, including client applications, backend services, infrastructure and settlement mechanisms.

The platform does not distinguish between product quality and security quality. A system that can be manipulated cannot be considered functionally correct.

# Primary Security Objectives

The security architecture is designed to achieve six primary objectives.

## Preserve Challenge Integrity

Challenge outcomes should reflect verified participant activity rather than software manipulation, operational mistakes or unauthorized intervention.

The platform should minimise opportunities for users to gain an unfair competitive advantage through modified clients, fabricated activity records or exploitation of business logic.

Whenever uncertainty exists, preserving competitive integrity takes precedence over convenience or processing speed.

---

## Protect Participant Assets

Financial value introduced through paid challenges must remain protected throughout the entire challenge lifecycle.

The long-term architectural objective is to ensure that participant funds cannot be redirected, withdrawn or redistributed outside the predefined settlement rules.

Financial authority no longer rests with the platform: settlement runs through deterministic smart contracts on mainnet, and the backend has no capability to move participant funds.

---

## Minimise Trust Requirements

Users should not be required to trust individual operators more than necessary.

Where practical, trust should be replaced by independently verifiable system behaviour.

This principle is realised in the non-custodial settlement, transparent challenge rules and publicly auditable smart contract logic.

Reducing required trust improves both platform resilience and long-term credibility.

---

## Protect User Privacy

Security and privacy are closely related but represent different objectives.

Security protects information from unauthorized access.

Privacy limits which information exists in the first place.

STRYNER therefore reduces security exposure by intentionally limiting the collection, transmission and storage of personal information.

Information that is never collected cannot become the subject of unauthorized disclosure.

---

## Maintain Service Availability

Challenge participation depends on predictable platform availability throughout multi-day events.

The architecture should therefore tolerate temporary failures of individual services without permanently affecting participant progress or settlement outcomes.

Whenever full functionality cannot be maintained, the preferred behaviour is graceful degradation rather than unpredictable operation.

Availability supports fairness by ensuring that all participants operate under consistent conditions.

---

## Enable Independent Verification

Platform decisions should be explainable and, wherever practical, independently verifiable.

Participants should be able to understand how challenge outcomes are determined, how rewards are calculated and why particular architectural constraints exist.

Transparency supports both security auditing and long-term confidence in the platform.

---

# Security Design Priorities

Not every security objective has equal importance in every situation.

When architectural trade-offs become necessary, STRYNER follows the following priority order:

1. Protection of participant assets.
2. Preservation of challenge integrity.
3. Protection of user privacy.
4. Correctness of settlement.
5. Availability of platform services.
6. Operational convenience.

This ordering reflects the belief that temporary inconvenience is preferable to incorrect challenge results or financial loss.

---

# Security Throughout the Platform Lifecycle

Security requirements evolve alongside the platform.

In the current phase, emphasis is placed on validating challenge mechanics, anti-abuse controls and operational reliability.

With smart-contract settlement now live, the focus is on deterministic financial execution, contract review and minimising custodial responsibilities.

Future security improvements are expected to strengthen existing controls rather than fundamentally alter the platform's underlying trust model.

---

# Scope of the Security Architecture

The following chapters describe how STRYNER approaches security from multiple perspectives.

Rather than focusing exclusively on technical controls, the document examines:

- the types of adversaries the platform expects,
- the assumptions upon which security depends,
- potential attack vectors,
- architectural mitigations,
- operational controls,
- and the residual risks that remain despite implemented safeguards.

Security is therefore presented as an architectural property emerging from multiple independent layers rather than from any single defensive mechanism.

---

# Conclusion

The objective of the STRYNER security architecture is not to promise absolute protection, but to build a platform in which failures remain contained, trust requirements are progressively reduced, and challenge outcomes remain fair, transparent and resilient.

These objectives establish the framework for the remainder of this section, beginning with the definition of the adversaries and threat scenarios against which the platform is designed to operate.

# 3.2 Adversary Model

## Purpose of This Chapter

A security architecture cannot be evaluated without first defining the capabilities of the adversaries it is intended to resist.

Different threat actors possess different objectives, resources and technical abilities. A platform designed to withstand accidental misuse may remain vulnerable to deliberate abuse, while a platform focused exclusively on external attackers may overlook risks originating from trusted infrastructure or legitimate users.

The purpose of this chapter is therefore to define the classes of adversaries considered during the design of STRYNER and to establish the assumptions that guide subsequent threat analysis.

The adversaries described below represent realistic threat scenarios rather than exhaustive possibilities.

---

# Security Philosophy

STRYNER is designed according to a conservative security principle:

**Every externally controlled component should be assumed capable of malicious behaviour unless its integrity can be independently verified.**

This assumption applies equally to client applications, network communication and user-controlled devices.

Rather than relying on declarations made by individual components, the platform attempts to validate important information through independent mechanisms wherever practical.

This philosophy reduces reliance on trust while improving the resilience of the overall architecture.

---

# Threat Classification

For the purposes of architectural analysis, adversaries are divided into several categories based on their level of control over the system.

Each category introduces different security considerations and requires different defensive strategies.

---

## Honest Participant

The majority of platform users are expected to interact with STRYNER according to its intended purpose.

These participants:

- follow challenge rules,
- use unmodified applications,
- submit legitimate activity,
- have no intention of manipulating challenge outcomes.

Although honest participants do not intentionally create security threats, the architecture must still protect them from operational failures, software defects and unfair behaviour by others.

Protecting honest users remains the primary objective of the platform.

---

## Opportunistic Cheater

Some participants may attempt to gain an advantage using techniques requiring little technical knowledge.

Examples include:

- manually entering activity,
- exploiting application workflows,
- abusing referral mechanisms,
- attempting multiple registrations,
- intentionally delaying synchronization.

These attacks generally rely on weaknesses in application logic rather than software exploitation.

The platform therefore attempts to minimise opportunities for low-effort abuse through validation rules and deterministic business logic.

---

## Technically Skilled Attacker

A smaller class of adversaries may possess software engineering knowledge sufficient to analyse, modify or automate parts of the client environment.

Potential capabilities include:

- reverse engineering mobile applications,
- intercepting network traffic,
- modifying application behaviour,
- scripting API requests,
- automating repeated actions,
- analysing public application binaries.

The architecture assumes such activities are possible and therefore avoids treating client-side behaviour as authoritative.

Any information originating from user-controlled software should be considered potentially manipulated until independently validated.

---

## Compromised Client Device

The platform assumes that some participant devices may become partially or completely compromised.

Possible scenarios include:

- rooted Android devices,
- modified operating systems,
- debugging frameworks,
- runtime instrumentation,
- malware,
- unofficial application distributions.

A compromised device cannot be assumed to enforce application behaviour correctly.

Consequently, client-side validation primarily serves to improve data quality and user privacy rather than acting as the platform's final security control.

---

## Network Adversary

Communication between components occurs across networks that should be considered inherently untrusted.

Potential network-level capabilities include:

- passive traffic observation,
- connection interruption,
- packet replay,
- DNS manipulation,
- man-in-the-middle attempts where transport security is absent.

For this reason, confidentiality, integrity and authentication should never depend on the assumption of a trusted network environment.

---

## Infrastructure Failure

Not every security incident originates from malicious behaviour.

Failures may arise through:

- infrastructure outages,
- software defects,
- database failures,
- cloud provider disruptions,
- deployment mistakes,
- operational errors.

These events are treated as security-relevant whenever they have the potential to influence challenge fairness, participant assets or settlement correctness.

The architecture therefore distinguishes between component failure and system failure, aiming to prevent local problems from propagating throughout the platform.

---

## External Dependency Failure

Several critical platform capabilities depend on external services beyond STRYNER's operational control.

Examples include:

- Telegram authentication,
- Android Health platform,
- Google Play distribution,
- TON blockchain consensus.

The architecture assumes these providers operate correctly under normal conditions while recognising that temporary disruption remains possible.

Where practical, the platform is designed to tolerate temporary dependency failures without compromising recorded challenge data.

---

## Platform Operator

The platform operator possesses administrative authority over infrastructure.

Rather than assuming perfect behaviour indefinitely, the long-term architecture seeks to progressively reduce the level of authority exercised by operational infrastructure.

This is realised in the deterministic smart-contract settlement now in place, where financial execution becomes governed by publicly auditable code rather than operational discretion.

The objective is not to eliminate operational responsibility but to reduce the amount of trust required from participants.

---

# Threats Outside Scope

Certain categories of attack remain outside the intended security scope of STRYNER.

These include:

- compromise of Telegram itself,
- compromise of Android operating system security mechanisms,
- compromise of TON consensus,
- physical theft of participant devices,
- theft of user wallet private keys,
- vulnerabilities introduced by unofficial third-party software outside the platform.

While such events may affect individual users, they cannot reasonably be mitigated by the STRYNER application architecture alone.

Users remain responsible for maintaining the security of their own devices, wallets and credentials.

---

# Adversary Assumptions

The security architecture therefore assumes that:

- client software may be modified,
- user devices may be compromised,
- network communication may be observed,
- infrastructure components may fail,
- legitimate users may intentionally abuse platform rules,
- external dependencies may experience temporary disruption.

At the same time, the architecture assumes that cryptographic primitives, authenticated communication protocols and blockchain consensus remain fundamentally trustworthy.

These assumptions define the boundary conditions under which the remaining security architecture has been designed.

# Conclusion

The purpose of the adversary model is not to predict every conceivable attack, but to establish a realistic framework for evaluating security decisions.

By explicitly identifying who the platform is designed to defend against—and equally important, which threats fall outside its intended scope—STRYNER can evaluate future security mechanisms against consistent assumptions rather than hypothetical worst-case scenarios.

The following chapter builds directly upon this model by examining the specific attack vectors that arise from these adversaries and the architectural controls designed to mitigate them.

# 3.3 Threat Model

## Purpose of This Chapter

The purpose of this chapter is to identify the principal threats considered during the design of STRYNER and to explain how the platform architecture reduces their likelihood or limits their impact.

No practical software platform can eliminate every possible attack. Instead, security engineering focuses on reducing realistic risks through layered controls, independent validation and architectural separation of responsibilities.

The threat model presented here is intended to guide future development, security reviews and smart contract audits by providing a common understanding of the risks against which the platform is designed.

---

# Threat Modelling Approach

STRYNER evaluates threats according to three questions:

- What could go wrong?
- What would be the impact if it happened?
- Which architectural mechanisms reduce that risk?

Rather than relying on a single security mechanism, the platform applies multiple independent controls across different architectural layers.

This approach assumes that individual safeguards may eventually fail and therefore avoids depending on any single defensive measure.

---

# Client Manipulation

Because client software executes on hardware controlled by the participant, it cannot be treated as an authoritative source of truth.

Potential attacks include:

- modification of application behaviour,
- bypassing local validation,
- automated API interaction,
- manipulation of synchronisation timing,
- altered application binaries.

The architecture therefore treats every client-originated value as potentially untrusted.

Server-side validation remains the authoritative source for challenge qualification.

Client-side filtering exists primarily to improve data quality and reduce unnecessary transmission of unverifiable records rather than to enforce platform security.

---

# Activity Data Manipulation

The value of the platform depends on the integrity of recorded activity.

Potential attacks include:

- manually created activity,
- fabricated health records,
- duplicated records,
- modified provenance information,
- attempts to replay previously accepted data.

To reduce these risks, STRYNER applies multiple validation stages.

Activity is filtered before transmission, independently validated by backend services and evaluated against challenge rules before affecting participant progress.

No individual validation stage is considered sufficient on its own.

---

# Business Logic Abuse

Not every attack targets software vulnerabilities.

Some participants may attempt to exploit legitimate application behaviour in unintended ways.

Examples include:

- abusing referral mechanisms,
- repeated registration attempts,
- challenge timing manipulation,
- intentionally delayed synchronisation,

- exploitation of edge-case workflows.

Business logic is therefore designed to remain deterministic and rule-based.

Where ambiguity exists, platform behaviour should remain predictable rather than dependent on manual operator decisions.

---

# Backend Compromise

The backend performs orchestration, validation and challenge management.

Its compromise would represent a significant operational event.

The architecture therefore attempts to minimise the authority exercised by backend services.

Financial authority sits with the deterministic smart contracts rather than with operational infrastructure; their behaviour is publicly auditable.

This reduces both operational risk and required participant trust.

---

# Smart Contract Risks

Smart contracts introduce a different category of security considerations.

Unlike traditional backend services, deployed contract logic cannot easily be modified.

Potential risks include:

- implementation defects,
- incorrect settlement logic,
- unexpected execution paths,
- permanently locked funds,
- incomplete state transitions.

For this reason, paid challenge settlement is intentionally postponed until the contract suite has completed implementation, testing and independent security review.

Security review is treated as a prerequisite for production financial operation rather than a post-launch improvement.

---

# Infrastructure Risks

STRYNER depends on several managed infrastructure providers.

Potential operational threats include:

- temporary service outages,
- deployment failures,
- database unavailability,

- networking failures,
- cloud provider disruption.

The architecture separates responsibilities across infrastructure components so that failure of one service does not automatically propagate throughout the entire platform.

Where possible, temporary failures delay processing rather than producing inconsistent challenge outcomes.

---

# External Dependency Risks

Several critical platform capabilities depend on external systems outside STRYNER's operational control.

These include:

- Telegram authentication,
- Android Health platform,
- Google Play distribution,
- TON blockchain.

Such dependencies cannot be completely eliminated.

Instead, the architecture attempts to minimise coupling and clearly define the responsibility of each dependency within the overall system.

Failures affecting external providers should not invalidate already verified participant activity whenever recovery remains possible.

---

# Operational Risks

Operational mistakes represent another important class of threats.

Examples include:

- configuration errors,
- deployment mistakes,
- incorrect environment configuration,
- accidental service interruption,
- human error during maintenance.

Operational procedures therefore aim to minimise manual intervention wherever deterministic system behaviour is achievable.

Reducing operational complexity improves both reliability and security.

---

# Residual Risk

Despite layered validation and architectural separation, certain risks cannot be completely eliminated.

Examples include:

- previously unknown software vulnerabilities,
- operating system defects,
- cryptographic implementation flaws,
- compromise of trusted external providers,
- undiscovered smart contract vulnerabilities,
- coordinated abuse techniques not previously observed.

Residual risk is considered an unavoidable characteristic of complex distributed systems rather than evidence of architectural failure.

The objective is therefore not absolute elimination of risk but continuous reduction through iterative improvement, monitoring and independent security review.

---

# Security Principles Applied

The threats described throughout this chapter are mitigated using a consistent set of engineering principles:

- independent verification,
- defence in depth,
- least privilege,
- deterministic processing,
- minimised trust,
- architectural separation,
- graceful failure,
- transparent settlement.

These principles are applied across multiple components rather than concentrated within any single layer.

---

# Conclusion

The STRYNER threat model recognises that security depends not on preventing every possible attack, but on ensuring that realistic attacks either fail outright, become economically unattractive or remain sufficiently contained that they cannot compromise challenge fairness or participant assets.

The following chapter examines the platform from another perspective by identifying its **attack surface**—the external interfaces through which these threats may interact with the system. That analysis provides the basis for the detailed security controls described in the chapters that follow.

# 3.4 Attack Surface

## Purpose of This Chapter

Every software system exposes interfaces through which users, external services or other systems interact with it.

Each exposed interface represents a potential attack surface.

The objective of this chapter is to identify those interfaces, explain why they exist and describe the architectural principles used to reduce unnecessary exposure.

Reducing attack surface is considered more effective than attempting to secure unnecessary functionality after it has been introduced.

---

# Security Philosophy

STRYNER follows a simple architectural principle:

**Components should expose only the minimum functionality required to perform their intended responsibility.**

Every additional interface increases operational complexity, introduces new failure scenarios and expands the number of possible attack vectors.

The platform therefore favours fewer, well-defined interfaces over broad system accessibility.

---

# User Interface

The primary public interface of STRYNER is the Telegram Mini App.

Users interact with the platform by:

- authenticating through Telegram,
- browsing challenges,
- joining competitions,
- viewing progress,
- connecting wallets,
- claiming rewards.

The Mini App intentionally performs no activity verification and has no direct access to health information.

Its role is limited to user interaction and presentation.

This separation reduces the consequences of interface-level vulnerabilities.

---

# Companion Application

The Android Companion App exposes a separate interface dedicated exclusively to activity synchronisation.

Its externally visible responsibilities are intentionally limited to:

- reading authorised step records,
- filtering activity,
- authenticating with backend services,
- transmitting verified daily totals.

The Companion App performs no challenge calculations and cannot determine challenge outcomes independently.

Because the application executes on user-controlled hardware, all data originating from it is treated as potentially untrusted until independently validated.

---

# Backend API

The backend represents the largest externally accessible system component.

It exposes authenticated endpoints required for:

- challenge management,
- activity synchronisation,
- account operations,
- progress retrieval,
- settlement preparation.

The backend intentionally avoids exposing unnecessary administrative functionality through public interfaces.

Administrative operations remain isolated from participant-facing APIs.

Every request is processed according to authentication, authorisation and business-rule validation before affecting persistent application state.

---

# Wallet Interface

Blockchain interaction occurs exclusively through the user's own wallet using the TON Connect protocol.

The platform never receives private keys, recovery phrases or signing authority.

Its interaction with the wallet is limited to requesting transaction approval and reading the resulting public information required for challenge participation.

This significantly reduces the attack surface associated with asset custody.

---

# Smart Contract Interface

Settlement contracts expose only the functions required for challenge execution.

These include operations such as:

- participant registration,
- settlement submission,
- reward claims,
- refund execution,
- contract closure.

Contract functionality should remain intentionally minimal.

Every additional callable function increases audit complexity and expands the number of possible execution paths.

Minimising public contract interfaces improves both security and long-term maintainability.

---

# Database Exposure

The database is not directly accessible by participants or external applications.

All persistent data access occurs through backend services implementing authentication, authorisation and business validation.

Direct database exposure is intentionally avoided.

This architectural separation reduces the likelihood that database-level vulnerabilities can be exploited through publicly accessible interfaces.

---

# External Service Interfaces

STRYNER communicates with several external systems.

These include:

- Telegram,
- Android Health platform,
- Google Play,
- TON blockchain,
- managed infrastructure providers.

Each integration represents an additional trust boundary.

For this reason, communication with external services is isolated behind dedicated interfaces rather than being distributed throughout the application.

Limiting integration points simplifies maintenance while reducing the impact of changes introduced by external providers.

---

# Administrative Interfaces

Operational interfaces used for deployment, monitoring and maintenance are intentionally separated from participant-facing services.

Administrative capabilities are not considered part of the public application surface.

Where possible, operational functions remain inaccessible through public APIs.

Reducing administrative exposure decreases the opportunities for privilege escalation and accidental operational misuse.

---

# Unnecessary Attack Surface

An equally important security decision concerns functionality that is intentionally absent.

STRYNER deliberately avoids exposing interfaces for:

- manual activity submission,
- direct database access,
- user-generated scripting,
- third-party plugin execution,
- arbitrary file uploads,
- public administrative endpoints,
- unnecessary health permissions.

Every omitted capability represents one less interface requiring long-term security maintenance.

The safest interface is often the one that does not exist.

---

# Surface Reduction Strategy

Attack surface is reduced through several architectural principles:

- single-purpose components,
- clearly defined responsibilities,
- authenticated communication,
- least privilege,
- minimal permissions,
- isolated infrastructure,
- deterministic interfaces,
- limited public endpoints.

These principles are applied consistently across every architectural layer.

Security therefore begins by limiting exposure before introducing defensive controls.

---

# Future Evolution

As the platform evolves, additional interfaces may become necessary.

Every new external interface should satisfy three questions before implementation:

- Is the interface necessary?
- Can an existing interface be reused?
- Does the security benefit justify the increased exposure?

Features that increase attack surface without providing measurable architectural value should generally be avoided.

This approach supports long-term maintainability while preserving the simplicity of the overall security model.

---

# Conclusion

The STRYNER attack surface is intentionally constrained through architectural design rather than relying solely on defensive technologies.

By limiting exposed functionality, separating responsibilities and minimising public interfaces, the platform reduces both the number of potential attack vectors and the operational effort required to secure them.

The following chapters examine the security mechanisms implemented within each architectural layer, beginning with the client environment and the protections applied to user-controlled devices.

# 3.5 Client Security

---

# Purpose of This Chapter

The client environment represents the least trusted component of the STRYNER architecture.

Unlike backend infrastructure or blockchain settlement, client software executes on hardware fully controlled by the participant.

Applications may therefore be modified, inspected, automated or executed within compromised operating environments.

The purpose of this chapter is to explain how STRYNER approaches client security under these assumptions and why no client-side mechanism is considered authoritative for challenge verification.

---

# Trust Model

The Companion App and Telegram Mini App are designed under a zero-trust assumption.

Neither application is expected to provide security through secrecy.

Application binaries can be extracted.

Network traffic can be inspected.

Application logic can be analysed.

Execution environments can be modified.

These realities are treated as expected characteristics of client software rather than exceptional attack scenarios.

For this reason, client applications should improve usability, privacy and efficiency but should never become the final authority for security-critical decisions.

---

# Client Responsibilities

Each client application performs a narrowly defined set of responsibilities.

The Telegram Mini App provides:

- user interaction,
- challenge participation,
- progress presentation,
- wallet connection,
- account management.

The Android Companion App provides:

- Health Connect integration,
- local activity filtering,
- daily step aggregation,
- authenticated synchronization.

Neither application determines challenge outcomes.

Neither application performs settlement.

Neither application has authority over participant funds.

---

# Local Validation

The Companion App performs several validation steps before transmitting activity.

These include:

- rejecting manually entered records,
- validating recording provenance,

- removing duplicated activity,
- aggregating verified daily totals.

These controls improve data quality while reducing unnecessary network traffic.

However, they are not treated as security boundaries.

Every accepted value is independently evaluated again after reaching backend infrastructure.

The platform therefore assumes that local validation may be bypassed without compromising overall system integrity.

---

# Protection Against Client Modification

Because client software executes within an environment controlled by the participant, modification cannot realistically be prevented.

Examples include:

- application patching,
- runtime instrumentation,
- debugging frameworks,
- rooted operating systems,
- emulator execution,
- custom firmware.

Rather than attempting to detect every possible modification technique, STRYNER reduces dependence on client integrity.

The backend evaluates received information according to independently defined validation rules instead of trusting the behaviour of the originating application.

---

# Authentication

Both client applications authenticate before accessing backend services.

Identity is established through trusted external providers rather than proprietary authentication mechanisms.

Telegram authenticates platform identity.

TON Connect authenticates wallet ownership.

Backend-issued session credentials authenticate application requests.

Authentication therefore represents layered verification rather than reliance on any single identity source.

---

# Secure Communication

All communication between client applications and backend infrastructure occurs through encrypted HTTPS connections.

Transport security protects transmitted information against passive observation and unauthorised modification while travelling across untrusted networks.

Encryption alone, however, does not establish trust in the correctness of transmitted data.

The platform therefore combines authenticated communication with independent server-side validation.

---

# Data Minimisation

Client applications intentionally process only the information required to perform their responsibilities.

The Companion App requests permission exclusively for step records.

It does not request access to:

- GPS location,
- heart rate,
- calories,
- sleep,
- blood pressure,
- oxygen saturation,
- exercise sessions.

Reducing accessible information improves user privacy while simultaneously reducing the consequences of client compromise.

---

# Failure Behaviour

Temporary client failures should not permanently affect challenge participation.

Examples include:

- application closure,
- device restart,
- temporary network loss,
- delayed synchronization.

The architecture therefore relies on the operating system's health platform as the authoritative source of recorded activity rather than on temporary application state.

When synchronization resumes, activity is re-evaluated from persistent health records instead of relying on locally cached values.

This behaviour improves reliability while reducing opportunities for replay-related inconsistencies.

---

# Privacy Considerations

Client applications intentionally avoid collecting information unrelated to challenge verification.

No behavioural analytics are embedded.

No advertising SDKs are integrated.

No user profiling is performed.

No persistent device identifiers are collected.

No background telemetry unrelated to synchronization is transmitted.

Privacy therefore results from limiting functionality rather than from restricting subsequent use of collected information.

---

# Security Limitations

Certain risks cannot be eliminated within the client environment.

Examples include:

- compromised operating systems,
- malicious accessibility services,
- hardware modification,
- malware,
- device theft.

Such scenarios fall outside the direct control of STRYNER.

The platform therefore focuses on ensuring that compromise of an individual client does not automatically compromise challenge integrity or participant assets.

This distinction reflects the broader architectural principle that security should depend upon independent validation rather than trusted endpoints.

---

# Future Evolution

Future client security improvements may include additional integrity verification mechanisms, improved anomaly detection and expanded platform support.

Such enhancements are intended to strengthen existing validation rather than replace the platform's fundamental zero-trust assumptions.

Security improvements should reduce risk without increasing unnecessary access to user data.

# Conclusion

Client applications represent the first point of interaction between participants and the STRYNER platform, but they are intentionally designed to exercise minimal authority over security-critical decisions.

By assuming that client software may eventually be modified or compromised, the architecture avoids depending on endpoint trust and instead relies upon independent verification performed across multiple system layers.

This approach improves resilience while preserving the privacy, simplicity and portability of the client environment.

# 3.6 Backend Security

## 3.6.1 Why a backend exists at all

Settlement is on-chain. A reasonable reader asks what is left for a server to do.

The answer is that **steps happen in the physical world**. No blockchain can observe someone walking. A contract can verify arithmetic, signatures and deadlines with certainty; it cannot verify effort. Any system that rewards real-world activity must therefore have a component that decides what happened — and that component runs off-chain.

This is not a temporary limitation to be engineered away in a later version. It is inherent to the category. What the smart-contract migration changed is not *who determines outcomes* but *who can move money*.

Stated plainly:

| | **Before migration** | **After migration** |
|---|---|---|
| Who evaluates activity | Backend | Backend |
| Who holds participant funds | Operator | Nobody — the contract |
| Who can settle early or late | Operator | Nobody — chain time |
| Who can withhold a refund | Operator | Nobody — refunds are pull-based |
| Who can withdraw the pool | Operator | No such function exists |

The backend remains authoritative over *facts*. It loses authority over *funds*.

## 3.6.2 The trust boundary

STRYNER is designed on the assumption that any single component may eventually fail, be compromised, or be modified by a determined user. Components are therefore treated differently according to what can be assumed about them.

**The Companion App is an untrusted client.** It runs on hardware the user controls and can be modified. Its on-device filtering — rejecting manually entered steps, unrecognised data sources, and

duplicate records — improves data quality and protects privacy by discarding unverifiable records before transmission. It is not a security control, and it is not treated as one.

**The network is untrusted.** All traffic is encrypted; contents are assumed observable.

**The backend is a trusted service** in the specific sense that it enforces challenge rules and validates incoming data independently. It is not trusted with participant funds, and the contract's guarantees hold whether or not the backend behaves honestly.

Every value the backend receives is validated again server-side, without reference to what the client claims about itself.

## 3.6.3 Authentication

Identity comes from Telegram. STRYNER stores no passwords and issues no credentials of its own, which removes an entire class of breach: there is no password database to leak.

Every request that changes state — joining a challenge, creating one, rotating a device pairing, initiating a transaction — carries a cryptographic proof of identity issued by Telegram and verified server-side on each call. Proofs are time-bounded, so a captured request cannot be replayed indefinitely.

Pairing between the Mini App and the Companion App uses a token that the participant can rotate at any time. Rotation immediately invalidates the previous pairing, so a device that is lost, sold or compromised can be cut off by the user without contacting support.

Read-only endpoints and endpoints that change state are separated deliberately: the authentication requirement is applied to the second group, and the boundary between them is part of the code review surface rather than a matter of convention.

## 3.6.4 Independent validation of activity data

The Companion App transmits a minimal payload: a user identifier, a date, a verified step total, and a synchronisation source marker. It does not transmit device identifiers, source application names, raw pre-filter counts, or operating system details.

The backend applies its own validation to every value it receives, and applies challenge rules — daily goals, completion criteria, ranking — server-side only. The client is never asked whether a day was completed; it is asked only what the health platform reported.

Anomaly detection operates on submitted data and flags implausible patterns for review before any value is settled. **The specific rules are not published.** A document describing exactly which patterns are flagged is a manual for evading them, and publishing it would weaken protection for every honest participant. This is the one area where transparency and security genuinely conflict, and the trade-off is made in favour of participants who are not trying to cheat.

## 3.6.5 Data minimisation as a security property

The most reliable protection for data is not collecting it.

- No analytics tooling of any kind: no tracking pixels, no behavioural analytics, no third-party telemetry
- No IP addresses, user agents or device identifiers written to logs

- Wallet addresses masked in operational logs
- Short log retention rather than indefinite archives
- Health data limited to step counts; distance, heart rate, location, exercise sessions and all other metrics are never requested at the permission level

A breach of the logging layer therefore exposes internal identifiers and little else. This is a deliberate design choice, not an accident of scale.

## 3.6.6 Settlement key management

The backend signs settlement results with a dedicated key. This key is worth describing precisely, because it is the highest-value secret in the system.

**What the key can do:** produce a settlement result that the contract will accept for one challenge, within a fixed time window.

**What the key cannot do:** move funds to an arbitrary address, alter a challenge's rules, settle outside the permitted window, prevent refunds, or affect any challenge other than the one a given signature names. The signed result is bound to a specific contract address, so a signature valid for one challenge is worthless against another.

The key is a signing key, not a spending key. It is held separately from any wallet and from application credentials.

## 3.6.7 The residual risk, stated plainly

A specification that hides its weaknesses is worth less than one that names them.

If the settlement key were compromised, an attacker **could not steal funds**. There is no path from a challenge contract to an attacker-chosen address; the operator's own share is computed on-chain from the settled pool rather than claimed, and every deadline is enforced by chain time.

An attacker **could** misdirect who among participants is paid — allocating a challenge's distributable pool to wallets that did not earn it, within otherwise correct totals. The contract verifies arithmetic, not merit; it has no independent knowledge of who walked.

Two properties bound this risk:

**Exposure is limited to one challenge.** Each challenge is an independent contract. A compromised settlement affects that challenge's pool and nothing else — not other challenges, not participant deposits elsewhere on the platform.

**Every settlement is permanently visible.** Settlement results are published on-chain and attributable. A wrong allocation cannot be quietly reversed or hidden; it becomes part of the public record the moment it is submitted.

This is **detection rather than prevention**, and it is stated as such. Eliminating it entirely would require the chain to independently observe physical activity, which no current design achieves. Reducing it further is a goal of later protocol versions; claiming it is already solved would be false.

### 3.6.8 Summary

The backend remains the authoritative source of truth about activity, because activity happens where no contract can see it. What the smart-contract migration removes is the operator's ability to hold participant funds, to control settlement timing, and to interfere with refunds.

Security therefore does not rest on trusting any single component. It rests on independent verification layers — device provenance, server-side revalidation, on-chain enforcement of the rules that govern money — each of which would have to fail simultaneously for an incorrect outcome to reach a participant's wallet.

# 3.7 Verification Security

## 3.7.1 Why verification exists

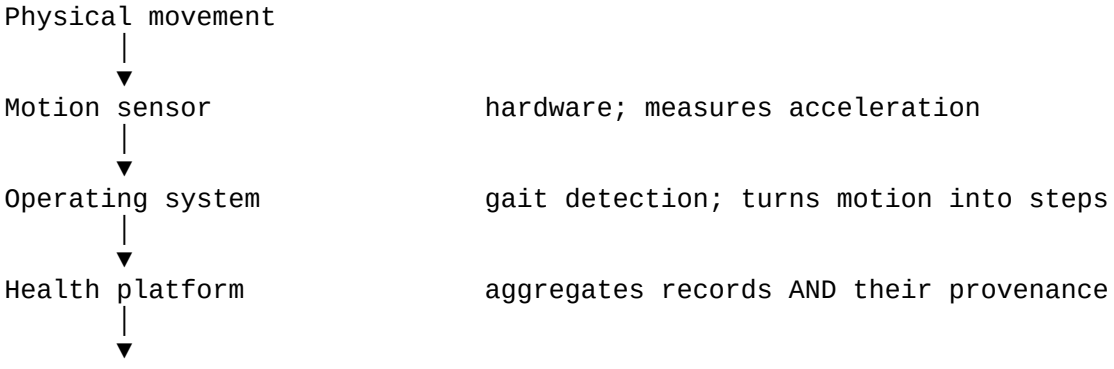
A commitment device is only as good as its measurement. If a participant can claim steps they did not take, every other guarantee in this document becomes decorative: the contract will settle honestly on dishonest input, the escrow will pay out correctly to the wrong person, and the cryptography will faithfully sign a lie.

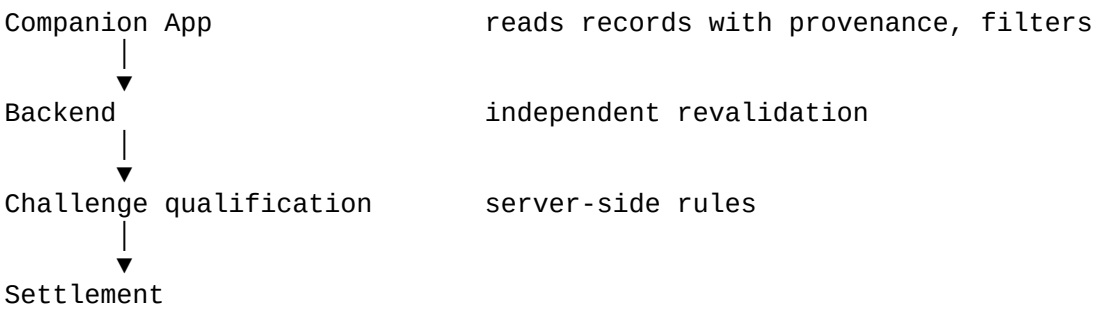
This makes verification the hardest problem in STRYNER — harder than the smart contracts, and the only part of the system that cannot be solved with cryptography alone.

The difficulty is structural. Financial data has a natural ledger: a transaction exists because a network agreed it exists. **Physical activity has no ledger.** Walking leaves no cryptographic trace. There is no authority that can attest, in a form a machine can check, that a particular person moved a particular distance on a particular day.

What exists instead is a chain of custody: a sequence of components, each of which observes something and records what it observed. Verification means understanding that chain — what each link can attest to, what it cannot, and where a claim can be substituted for an observation.

## 3.7.2 The chain of custody






Each link contributes something the next cannot reconstruct.

**The sensor** measures acceleration. It cannot distinguish walking from any other motion with a similar signature; that is not its job.

**The operating system** applies gait detection, converting raw motion into step events. This is where most incidental movement is filtered out — a phone in a car, a hand gesture, a bumpy road. The OS-level step counter is substantially more trustworthy than raw accelerometer data, and it is not something an application can replicate as well.

**The health platform** is the critical link, and not for the reason most people assume. Its value is not that it stores step counts — any database could. Its value is that it stores **who wrote each record and how it was produced**.

**The Companion App** reads records together with that metadata and applies filtering before anything leaves the device.

**The backend** revalidates independently, treating the app's output as a claim rather than a fact.

# 3.7.3 Provenance is what makes verification possible

A step count on its own is not evidence. It is a number, and any application with write access to a health platform can write any number it likes.

What turns a number into something checkable is the metadata recorded alongside it:

- **How the record was produced** — automatically by the device, actively recorded by a fitness application, or entered by hand
- **Which application wrote it** — identified by the platform, not self-declared by the writer

This distinction carries the entire verification model. Without provenance, filtering is impossible and every value must be taken on trust. With provenance, records that were typed rather than measured, or written by unrecognised software, can be identified and discarded before they influence any outcome.

This is why STRYNER reads from a system health platform rather than counting steps itself. An application that measures its own steps is asserting a number. An application that reads from the platform is reading an attested record — and can see the attestation.

## 3.7.4 Automatic sources only

STRYNER accepts step records produced in two ways:

- **Automatically recorded by the device**, without user involvement
- **Actively recorded by a fitness application** during a tracked activity

Records produced any other way are rejected. In practice the significant case is **manual entry**: most health platforms allow a user to type a step count directly, and such records are indistinguishable from measured ones once the number is stored — except in their provenance metadata, which is exactly where they are caught.

This rejection is categorical rather than heuristic. A manually entered record is not scored, weighted or discounted; it does not enter the pipeline at all.

## 3.7.5 Recording provenance

Beyond how a record was made, STRYNER checks **what made it**.

Accepted records must originate from the system health platform itself or from a recognised fitness provider. Records written by unrecognised applications are discarded.

The reasoning is direct: on a platform that permits third-party writes, an arbitrary application can write arbitrary values with correct-looking metadata. Restricting accepted sources to a known set narrows this from "anything installed on the device" to "a specific, reviewable list".

The list is a maintained asset rather than a fixed constant. Adding a provider is a deliberate decision, and the criteria are described in 3.7.10.

## 3.7.6 Rejected records never leave the device

Records that fail provenance checks are discarded **on the device**. They are not transmitted, not stored, and not logged.

This has two distinct benefits that are worth separating, because they are often conflated:

**Integrity.** Invalid data cannot influence an outcome if it never arrives.

**Privacy.** The backend never receives information about which applications a participant has installed, what other activity they recorded, or what data they chose not to sync. Rejected records would reveal all three.

The transmitted payload is deliberately minimal: an identifier, a date, a verified total, and a marker indicating the synchronisation path. Source application names, raw pre-filter counts, device identifiers and operating system details are not included.

## 3.7.7 Duplicate detection

A single walk can be recorded several times.

Modern devices frequently run more than one application with health platform access — a manufacturer's fitness app, a third-party tracker, a smartwatch companion. Each may write its own record of the same physical movement. Naively summing every record would multiply one walk into several, awarding a participant for activity they performed once.

This is not an attack. It is the normal state of a well-used device, and a verification system that ignores it produces inflated results for ordinary honest users — which is worse than an occasional cheat, because it happens constantly and silently.

STRYNER resolves overlapping records by counting only the primary system source when multiple sources describe the same period. A step is counted once regardless of how many applications observed it.

## 3.7.8 Server-side revalidation

**The Companion App is an untrusted client.** It runs on hardware the participant controls, and it can be modified.

On-device filtering therefore serves data quality and privacy — it discards unverifiable records early and keeps them off the network — but it is **not a security control**, and the architecture does not treat it as one.

Every value the backend receives is validated again, server-side, without reference to what the client asserts about itself. The client is never asked whether a record was valid, whether a goal was met, or whether a day was completed. It is asked only what the health platform reported, and the server decides what that means.

Beyond structural validation, submitted data is checked for implausible patterns before it can influence a settlement. Flagged results are held for review rather than settled automatically.

**The specific detection rules are not published.** A document describing precisely which patterns are flagged is a manual for evading them. This is the single point where transparency and protection genuinely conflict, and the trade-off is resolved in favour of participants who are not attempting to cheat.

## 3.7.9 Challenge qualification

Verification produces a daily total. Qualification turns that total into a challenge outcome, and it happens entirely server-side.

**Daily goal.** A day counts as completed when the verified total meets the goal fixed when the challenge was created. The goal is not adjustable afterwards, by the participant or by the operator.

**Time zone.** A participant's time zone is fixed at the moment they join and stored with their entry. Days are evaluated against that fixed zone for the whole challenge.

This is deliberately less forgiving than evaluating against a participant's current location, which would suit travellers better. The reason is determinism: settlement must be reproducible byte for byte, because the result is committed to a cryptographic root that anyone can verify. A rule that depends on where a participant happened to be at evaluation time cannot be recomputed reliably, and therefore cannot be part of a verifiable settlement.

**Verification window.** Settlement cannot occur until a fixed period after a challenge ends, long enough for the last local day to have closed anywhere in the world and for synchronisation to complete. This window is enforced on-chain and cannot be shortened by the operator.

## 3.7.10 Provider independence

Nothing in this model is specific to one operating system or one health platform. The pipeline is defined by properties a source must have, not by the identity of the source.

A data source can be integrated if it provides:

1. **Provenance metadata** — how a record was produced and what produced it
2. **A distinction between measured and entered data** — without this, manual entry cannot be rejected
3. **Stable record identity** — sufficient to detect the same activity reported twice
4. **Read access without write dependence** — STRYNER reads; it does not need to write

Sources that satisfy these can be added without altering the verification model. Sources that do not — for example, one that reports totals with no indication of origin — cannot be made trustworthy by processing them more carefully downstream, because the information required simply is not present.

This is why the requirements are stated as properties rather than as a list of supported apps. The list will change; the requirements should not.

## 3.7.11 Limitations

A verification chapter that claims completeness would not deserve to be believed. The following are not solved.

**Sensor-level manipulation.** The chain begins with a physical sensor, and a sensor reports what it detects. Deliberate physical manipulation of a device can produce motion that a step counter interprets as walking. Operating-system gait detection filters a substantial part of this, and anomaly detection catches patterns that survive it, but neither is a proof.

**Device transfer.** A participant who gives their phone to someone else has not broken any cryptographic property. The system verifies that a device recorded activity; it does not verify who was carrying it.

**Compromised devices.** On a device where the platform's own protections have been removed, an attacker may be able to write records that appear to originate from a recognised source. On-device filtering cannot detect this, because filtering trusts the platform's metadata and the platform itself has been subverted. Server-side anomaly detection is the backstop here, not the filter.

**The measurement is a proxy.** Steps are a proxy for effort, and any proxy can be satisfied without the underlying thing it stands for.

None of these are unique to STRYNER. They apply to every activity-based system that reads from consumer devices. They are stated here because the alternative — implying they are handled — would be false.

### 3.7.12 Residual risk

The honest summary is that verification is **layered mitigation, not proof**.

Three properties bound the exposure.

**Cost against reward.** Value at stake in any single challenge is capped by that challenge's pool, which is a function of its entry amount and participant count. Sustained physical manipulation of a device costs time and attention. For challenges of the size STRYNER is designed around, the effort required to cheat reliably is disproportionate to what can be won — and unlike a financial exploit, it cannot be automated once and replayed at scale, because each fraudulent result must be produced continuously across the whole challenge period.

**Detection before settlement.** Anomalous results are held for review rather than settled automatically. Detection is imperfect, but it operates before funds move rather than after.

**Bounded blast radius.** Each challenge is an independent contract. A verification failure affects the pool of the challenge it occurred in — not other challenges, and not participant deposits elsewhere on the platform.

What would improve this in later versions is additional independent sources: a second device or provider corroborating the same activity, so that manipulation must be sustained across systems that do not share a failure mode. This is a direction, not a claim about the current implementation.

### 3.7.13 Summary

STRYNER does not measure steps. It reads records that a device's own health platform has already attested to, together with the metadata describing how those records came to exist — and it uses that metadata to distinguish measurement from assertion.

That distinction is the core of the product. The smart contracts determine who can move money; the verification pipeline determines who deserves it. Neither is sufficient alone, and of the two, the second is the one that cannot be replaced by cryptography.

## 3.8 Wallet Security

### 3.8.1 Two signing systems, only one of which STRYNER holds

Readers of blockchain documentation routinely conflate two different things called "the key". Separating them is the first step in understanding what STRYNER can and cannot do to a participant's funds.

| | Participant's wallet key | Settlement (oracle) key |
|---|---|---|
| Held by | The participant, in their own wallet | STRYNER, in backend infrastructure |
| Can spend funds | Yes — it is the only thing that can | **No** |
| Can authorise a challenge entry | Yes | No |
| Can determine challenge outcomes | No | Yes, within fixed limits |
| Visible to STRYNER | **Never** | Yes, by definition |
| Recoverable if lost | Only by the participant | Rotatable by deploying a new factory |

The asymmetry is the point. STRYNER holds a key that decides *who is owed what* and holds no key that can *move anything*. The participant holds a key that can move their own funds and holds no key that decides outcomes.

Neither party holds both. That is the entire security posture of the wallet layer, and everything below is detail.

## 3.8.2 What STRYNER never sees

The following never reach STRYNER's systems, in any form, at any point:

- Private keys
- Seed phrases or recovery phrases
- Wallet passwords, PINs or biometric data
- Wallet-internal state
- Balances of assets unrelated to STRYNER
- Transaction history outside STRYNER's own contracts
- Which other applications a participant has connected

STRYNER does not create wallets, derive keys, store keys, escrow keys, or offer key recovery. There is no code path that could transmit a key even in error, because no interface exists that accepts one.

What STRYNER does see is the participant's **public address** and the transactions involving STRYNER's own contracts. Both are public blockchain data, visible to anyone with a block explorer. Connecting a wallet does not disclose anything that was previously private.

This is worth stating precisely because "connect your wallet" sounds like a handover and is not one. **A connection is a permission to ask, not a transfer of control.**

## 3.8.3 The authorisation model

Blockchain interaction occurs exclusively through the participant's own wallet using the TON Connect protocol. A connection grants STRYNER exactly two capabilities:

1. **Read the public address**, so the platform knows which wallet is participating
2. **Request approval for a specific transaction**, which the participant may accept or refuse

It does not grant the ability to move funds, to sign on the participant's behalf, or to act while the participant is absent.

**There is no standing authorisation.** STRYNER requests no delegated spending permission, no pre-approval covering future transactions, and no allowance that persists beyond a single action. Every transaction that moves value — joining a challenge, creating one, claiming a reward, withdrawing a refund — is a separate request that the participant approves individually in their own wallet.

This deserves emphasis because a widespread pattern in other ecosystems works differently: an application requests a broad, open-ended spending permission once, and thereafter moves funds without further prompts. Such permissions are a recurring source of loss, because they persist long after the user has forgotten granting them and remain exploitable if the approved contract is later compromised.

STRYNER requests no such permission. If the platform disappeared tomorrow, no lingering authority over any participant's wallet would remain.

## 3.8.4 What a signature actually commits to

When a participant approves a STRYNER transaction, they are not signing a general endorsement. They are signing a specific message with a specific destination, amount and payload, which their wallet displays before signing.

Concretely, a challenge entry commits to sending a stated amount to a stated contract address. Nothing more is authorised by that signature: it does not authorise a second entry, a different amount, or a transfer to any other address.

Participants are encouraged to read what their wallet displays rather than approving reflexively. This is standard advice, and it is repeated here because it is the single most effective protection available at the wallet layer, and it is one STRYNER cannot apply on a participant's behalf.

To make that verification practical, the platform keeps the number of transactions a participant must approve deliberately small, and publishes the contract addresses involved so a cautious participant can check the destination independently.

## 3.8.5 Replay protection

A replayed transaction is one captured and submitted again to produce a second effect from a single authorisation. STRYNER's exposure is bounded at two independent layers.

**At the wallet layer**, TON wallet contracts include a sequence number that increments with each executed message. A message carrying a consumed sequence number is rejected by the wallet itself. This protection belongs to the participant's wallet, not to STRYNER, and applies to every application they use.

**At the contract layer**, STRYNER's own protections are independent of the wallet's:

- **Settlement results are bound to one deployment.** The signed payload contains the address of the specific challenge contract it settles, along with a domain identifier and a network identifier. A settlement signed for one challenge is rejected by every other challenge, on either network. This is verified by test on a live chain, not merely asserted.

- **The state machine is one-way.** A challenge that has settled rejects a second settlement regardless of signature validity — the state guard triggers before signature checking is even relevant.
- **Each reward can be claimed once.** The contract records claimed entitlements and rejects a repeat, verified on-chain.
- **Each refund can be taken once.** Withdrawing removes the participant's entry, so a second attempt finds nothing to refund.

The layers are deliberately independent. A weakness in one does not compromise the other, and neither relies on the participant's wallet behaving correctly for STRYNER's guarantees to hold.

## 3.8.6 Wallet rotation, and the honest cost of non-custody

A participant may connect a different wallet at any time. Future challenges will be entered from the newly connected address, and the platform imposes no restriction on changing wallets.

**However**, entitlements are bound to the address that entered a challenge. A reward earned by wallet A is claimable by wallet A. Connecting wallet B does not transfer that claim.

This is a direct consequence of how the settlement works: each entitlement is committed cryptographically to a specific address before any claim occurs, and the contract reconstructs the claim from the address of whoever is asking. An address that was not part of the settlement cannot produce a valid claim, and no operator action can change that after the fact.

The practical implication must be stated without softening:

> **If a participant permanently loses access to the wallet they entered a challenge with, STRYNER cannot recover their funds. Neither can anyone else. There is no administrative override, no support-desk restoration, and no key held anywhere that could authorise a transfer to a replacement address.**

This is not an oversight. It is what non-custodial means. The same architecture that prevents the operator from seizing participant funds also prevents the operator from returning funds to someone who has lost their key. A system offering both properties simultaneously does not exist: any mechanism that could restore access to a lost wallet is by definition a mechanism that could take funds from a live one.

Participants should therefore treat wallet backup with the same seriousness they would apply to any other self-custodied asset, and should consider entering challenges from a wallet whose recovery phrase they have securely recorded.

## 3.8.7 Risks at the wallet layer that STRYNER cannot mitigate

The wallet is outside STRYNER's trust boundary in both directions. The platform cannot compromise it, and cannot protect it.

**Compromised devices.** A device with malware capable of reading wallet state or manipulating approval dialogs can act as the participant. No property of STRYNER's design changes this.

**Phishing and impersonation.** An attacker may present a convincing copy of the interface and request approval for a transaction to an address they control. The wallet will faithfully display what it is asked to sign; the participant is the only party positioned to notice that the destination is wrong.

**Malicious or compromised wallet software.** A wallet application that misrepresents what it is signing defeats every protection above it. Participants should obtain wallet software from its official source.

**Approval fatigue.** A participant who habitually approves without reading has effectively delegated spending authority to whatever their screen happens to show.

What the platform does to reduce exposure, within its reach: request as few transactions as possible, keep each one simple enough to read, publish contract addresses so destinations can be verified independently, and avoid any interaction pattern that trains participants to approve without looking.

None of these is a substitute for the participant checking what they sign.

## 3.8.8 The current phase, stated honestly

STRYNER's settlement sits in per-challenge contracts, with no operator custody at any point. This is live on mainnet and is described in 3.9.

The contracts have been tested extensively, including the full lifecycle executed with real value on live mainnet. What they do not yet have is an independent security review; until that review is complete, participants rely in part on code that is well-tested but not independently audited — not on a contract that makes misbehaviour impossible.

This is stated plainly rather than blurred, because the distinction between "non-custodial" and "intending to become non-custodial" is exactly the sort of thing a whitepaper should not leave ambiguous. **The settlement contracts are complete and live on mainnet; their independent review is the outstanding item, treated as a residual risk in 3.9.13.**

What has never been custodial, at any stage, is the participant's own wallet key. The custody question concerns pooled challenge funds, not keys, and no phase of STRYNER's development has involved the platform holding participant keys.

## 3.8.9 Summary

The wallet layer rests on a single structural property: **STRYNER holds no key capable of moving participant funds, and the participant holds no key capable of determining challenge outcomes.**

From this follow the specific guarantees. Connecting a wallet discloses nothing that was not already public. Every value-moving action requires individual approval, and no standing authority is requested or retained. Signatures commit to specific destinations and amounts, protected against replay at both the wallet and contract layers.

And from the same property follows the cost, which is stated rather than minimised: nobody can recover a lost wallet, because nobody has the authority that recovery would require.

# 3.9 Smart Contract Security

## 3.9.1 What a contract can and cannot guarantee

Smart contracts are frequently described as if they made systems trustworthy. They do something narrower and more useful: they make certain rules **unbreakable by anyone, including their author**.

A contract executes deterministically. Given the same state and the same message, it produces the same result, and no party — operator, participant or observer — can persuade it otherwise. What it cannot do is know whether the input it received was true.

This distinction carries the whole chapter. STRYNER's contracts enforce arithmetic, deadlines, and the impossibility of certain actions. They do not, and cannot, verify that anyone walked. That question is answered off-chain, by the verification pipeline described in 3.7, and the contract accepts its answer.

The contracts therefore make a specific and bounded promise:

> Whatever the settlement says, the money moves according to rules fixed before anyone joined, on a schedule nobody can alter, to addresses nobody can substitute — and if the settlement never arrives, everyone gets their entry back without needing the operator's cooperation.

That is a smaller claim than "trustless fitness", and it is one the architecture can actually keep.

## 3.9.2 One contract per challenge

Each challenge is deployed as its own independent contract, rather than as a record inside a single pooled contract holding every participant's funds.

The pooled design is more efficient. It costs less to deploy, consumes less blockchain storage, and is simpler to index. It was rejected for one reason: **a defect in a pooled contract is a defect affecting every participant simultaneously.**

With per-challenge contracts, the properties change qualitatively:

| | Pooled contract | Per-challenge contracts |
|---|---|---|
| Funds at risk from one defect | All participants, all challenges | One challenge's participants |
| A stuck challenge affects | Everyone | Only its own participants |
| Configuration changes affect | All existing challenges | Only challenges deployed afterwards |
| Failure of one settlement | Systemic | Contained |

The cost is real and is not hidden: every challenge pays its own blockchain storage, and deployment is a per-challenge expense rather than a one-time one. This is funded transparently — participants see a contract reserve as a separate line item before joining, and any unused portion is returned to participants rather than retained.

The trade is deliberate. Efficiency is paid for in currency; isolation is paid for in blast radius, and blast radius cannot be refunded.

## 3.9.3 The factory, and what a challenge creator controls

Challenges are created by the community, not by the operator. Any participant can create one.

This raises an obvious question: if anyone can deploy a challenge, what stops them from deploying one that looks official but pays out to them?

The answer is that a creator does not supply the parameters that would allow it. The factory supplies those itself.

**What the creator chooses** — the shape of their challenge:

- Entry amount
- Participant capacity, from a fixed set of permitted sizes
- Duration and daily goal
- Start and end dates

**What the factory injects** — the parameters that govern money and trust:

- The settlement signing key
- Payout addresses for treasury, platform and community vault
- The distribution split
- Every time window, including the settlement and refund periods
- The contract reserve and infrastructure fee rate

A creator cannot substitute their own settlement key, redirect payouts, or alter the split — **not because the contract rejects such an attempt, but because those fields do not exist in the message a creator sends.** There is no parameter to tamper with.

The factory validates what the creator does supply. Entry amounts below a floor are rejected, as are capacities outside the permitted set and dates that are already in the past or inconsistent with each other.

**Administrative scope.** The factory administrator can change the injected parameters — but only for challenges deployed *afterwards*. There is no message, from any sender, that reaches an already-deployed challenge and alters its configuration. A challenge's rules are fixed in its own state at the moment it is created, and remain fixed for its entire life.

This means a participant evaluating a challenge is evaluating something permanent. What they read when they join is what will execute.

## 3.9.4 Immutable rules

The claim that a challenge's configuration cannot change deserves a precise mechanical explanation rather than an assurance.

On TON, a contract's address is derived from its initial state — its code together with its starting data. This has a consequence that is easy to state and impossible to circumvent: **a contract with different configuration is a different contract, at a different address.**

There is no operation that modifies a deployed challenge's parameters, because such an operation is not merely forbidden by the code; it is incoherent with how addresses work. A participant who

sends funds to a particular address is sending them to a particular set of rules. Those rules cannot be swapped underneath them.

Everything that governs the outcome is fixed at that moment: entry amount, capacity, duration, daily goal, all deadlines, the distribution split, the settlement key, and every payout destination.

## 3.9.5 There is no administrative withdrawal

The contract exposes no function permitting the operator, the challenge creator, or any other privileged party to withdraw participant funds.

This is worth distinguishing from a weaker and more common design. Many contracts include an emergency withdrawal restricted to an owner address, or gated behind a multi-signature wallet, or subject to a timelock. Each is an improvement on unrestricted access, and each still means the capability exists — protected by a key, a process, or a delay, all of which can fail.

**STRYNER's challenge contracts have no such function at all.** There is nothing to protect, nothing to gate, and no key whose compromise would grant access. A reviewer verifying this claim does not need to evaluate the strength of a protection mechanism; they need to confirm the absence of a code path, which is a substantially easier thing to verify.

The addresses the contract can send funds to are fixed at deployment, and each is reachable only through a specific rule: participants through claims and refunds, treasury and platform through their calculated share, and the community vault through closure. There is no path to an address supplied at runtime.

## 3.9.6 Deadlines enforced by the chain

Every time-based rule is enforced by blockchain time rather than by the backend.

**Registration** closes at a fixed moment. Entries after it are rejected by the contract, not by a server that might be misconfigured or malicious.

**Settlement cannot occur early.** A verification window opens only after a fixed period following the challenge's end — long enough for the last local day to have closed anywhere on Earth, plus a margin for activity data to synchronise. The operator cannot shorten this. A settlement submitted before the window opens is rejected regardless of whether its signature is valid.

**Settlement cannot occur late.** Beyond a fixed deadline, the contract stops accepting settlements entirely and the failure path becomes available to anyone.

**Claim and refund periods** run for fixed durations from the events that start them, and cannot be extended or curtailed.

The significance is that these guarantees do not depend on the operator's continued good behaviour, or continued existence. A backend that is compromised, misconfigured or simply switched off cannot settle a challenge outside its window, because the contract measures time itself.

## 3.9.7 Pull-based claims

When a challenge settles, the contract records what each participant is owed. It does not attempt to send those amounts.

This is a deliberate inversion of the intuitive design, in which settlement distributes rewards automatically.

Automatic distribution fails in ways that are difficult to recover from. A single participant whose wallet cannot accept a transfer can stall a distribution loop. The number of transfers a single transaction may emit is bounded, so a large challenge cannot be distributed in one operation regardless of how it is written. And a distribution that fails midway leaves the contract in a state where some participants have been paid and others have not, with no obvious way to determine which.

With pull-based claims, each participant's withdrawal is an independent transaction. One participant's failure to claim, or their wallet's inability to receive, affects nobody else. There is no queue, no ordering, and no partial state to reconcile.

Each entitlement can be claimed exactly once. The contract records claims as they are made and rejects repeats — a property proven both in the test suite and on a live network, where a second claim attempt was correctly rejected after the first succeeded.

## 3.9.8 The refund state

This is the guarantee that matters most, because it is the one that protects participants when everything else has gone wrong.

If a valid settlement does not arrive before the deadline, **anyone** may move the challenge into its refund state. Not the operator, not a designated party — any address, including a participant's own.

Once in that state:

- Every participant may withdraw **their full entry amount**. Not a proportional share, not reduced by fees, not subject to any minimum.
- Withdrawal is a pull: the participant acts, and no other party is required or able to interfere.
- The withdrawal window is long, measured in months rather than days.

After that window, remaining deposits are pushed out in batches by a permissionless operation — the mechanism of last resort for participants who never returned. Anyone may run it; there is no reward for doing so, and a call that would pay nobody is rejected, so the operation cannot be used to consume resources without effect.

Both stages have been executed on a live network: a challenge deliberately left unsettled was moved to its refund state by an ordinary wallet, one participant's full entry was withdrawn by pull, and in a separate challenge a participant who never returned was paid by the sweep.

**The operator is not required at any point in this path.** That is the entire design intent: a participant's ability to recover their money must not depend on the continued existence, competence or honesty of the platform.

## 3.9.9 Treasury separation and the operator's share

The operator's revenue is separated from participant funds structurally rather than by policy.

**Three distinct addresses** receive value, and none of them is a general-purpose operator wallet: a treasury address, a platform address, and a community vault. Each has one defined role, and each is fixed at deployment.

**The operator's share is computed, not claimed.** When a challenge settles, the treasury and platform amounts are calculated on-chain from the settled pool according to the fixed split, and paid directly. They do not pass through the settlement data the operator submits, which means the operator cannot inflate them by submitting different figures. The arithmetic is the contract's, not the backend's.

**Unallocatable value goes to participants or to the community vault, never to the operator.** This principle is applied consistently:

- If nobody completes a challenge, the completion share goes to the participants who progressed furthest — not to treasury
- Referral allowances left unspent return to the participant pool
- Rewards left unclaimed after the claim period go to the community vault
- Contract reserve that was not needed goes to the community vault

The last two deserve comment. Unclaimed rewards are the classic case where a platform quietly benefits from user inattention, and unused reserve is the classic case where a platform benefits from over-estimating costs. Routing both away from the operator removes the incentive to design for either.

## 3.9.10 Blast radius

Isolation is easier to assess as a set of concrete questions.

**If one challenge contract has a defect, what is at risk?** That challenge's deposits. Other challenges are separate contracts holding separate funds, with no shared state and no shared balance.

**If the settlement key is compromised, what is at risk?** The allocation of one challenge's distributable pool at a time, within the fixed split. Not participant deposits, which are returned before any distribution; not other challenges, since each signature is bound to one contract address; and not the operator's ability to be detected, since every settlement is published on-chain.

**If the backend is compromised entirely, what is at risk?** The same as above, plus the availability of the service. Not custody, because no key capable of moving participant funds exists in that infrastructure.

**If the operator disappears?** Nothing is lost. Challenges awaiting settlement enter their refund state and participants withdraw in full, by a path that requires no operator involvement.

**If the factory is compromised?** Future challenges could be deployed with hostile parameters. Existing challenges are unaffected — their configuration is fixed in their own state, and no message from the factory can alter it.

## 3.9.11 Upgrade philosophy

**The challenge contracts are not upgradeable.** This is a deliberate refusal, not an omission.

An upgradeable contract is a contract whose rules can change after participants have committed funds to them. Whatever governance surrounds that capability — a multisig, a timelock, a DAO vote — the participant's position is the same: they are trusting a future decision, not a present guarantee. Upgradeability and immutability are mutually exclusive, and STRYNER's central promise is the second.

Improvements are therefore delivered by deploying a **new factory**, which produces new challenges under new rules. Existing challenges continue running to completion under the rules they were created with. Nobody's terms change mid-challenge, and nobody is migrated without consent.

This has a useful side effect. Rotating the settlement key — replacing a key that may have been exposed — requires no special mechanism and no upgrade path. A new factory is deployed with the new key; challenges created afterwards use it, and existing challenges finish under the old one, which remains valid only for them.

The cost is that improvements reach only new challenges, and that multiple factory versions may coexist. That is accepted as the price of the guarantee.

## 3.9.12 What has been verified, and what that does and does not mean

The contract suite is covered by an automated test suite exercising every handler, including deliberate attempts to break each guarantee. Beyond the happy path, the tests confirm that:

- A settlement correctly signed for a *different* challenge is rejected
- A participant cannot claim another participant's entitlement, nor inflate their own
- A second claim on the same entitlement is rejected
- A second refund is rejected
- Settlement outside its permitted window is rejected
- An unrelated wallet can trigger the refund state and run the sweep
- A partially completed sweep resumes correctly
- Closure is refused if the contract's internal accounting does not balance

Beyond the emulator, **every path has been executed on a live network** — including the failure paths, which are usually the least tested part of any system because they are inconvenient to reach. Challenges were created, joined, settled from the backend, claimed, and closed; separate challenges were deliberately abandoned to verify that expiry, pull refunds and the sweep all work under real conditions, with real timing and real fees.

**What this does not mean.** Testing demonstrates the presence of correct behaviour in the cases tested. It does not demonstrate the absence of defects. A test suite is written by the same people who wrote the code, and shares their blind spots. On-chain execution proves the paths work; it does not prove there is no path nobody thought to try.

That is what independent review is for, and it has not yet happened.

## 3.9.13 Security review

**An independent security review of the final contract code has not yet been performed.** It is the single most important item still outstanding before the contracts can be considered production-hardened.

This is stated without qualification because the alternative — implying that thorough testing is equivalent to review — is the specific misrepresentation that precedes most contract exploits.

The review will target a set of stated invariants: that contract balance always covers what is owed; that no path moves funds to an address not fixed at deployment; that entitlements can never exceed what settlement permitted; that each claim and each refund happens at most once; that state transitions are one-way; and that closure occurs only when the accounting balances.

Areas flagged internally as deserving particular attention include the handling of transfers that bounce back from unreachable addresses, the binding of settlement signatures against replay, and the sizing of the operational reserve — the last being a parameter derived from measurement rather than proof, and one that must be re-measured against the final implementation before it is fixed.

**Paid participation is live; the independent review is not.** This document states that plainly rather than implying the contracts have been audited: they have been tested across the full lifecycle on mainnet, but not independently reviewed.

## 3.9.14 Residual risk

**Contract defects.** The most significant remaining risk is a defect in the contracts themselves. Immutability, which protects participants from rule changes, also means a defect cannot be patched in a deployed challenge. Mitigations are isolation — one challenge's defect does not reach another — and review before funds are at stake. Neither is a guarantee.

**Settlement discretion.** As described in 3.6, a compromised settlement key can misdirect who among participants is paid, within correct totals. The contract enforces the arithmetic; it has no independent knowledge of who earned what. Exposure is bounded to one challenge's distributable pool, and every settlement is permanently visible on-chain.

**Reserve sizing.** Each contract carries a reserve to fund its own storage and execution over its lifetime. That figure is derived from measurement of a benchmark, and while it includes margin, an underestimate would leave a contract unable to complete its own closure. Re-measurement against the production implementation is an outstanding item.

**Platform risk.** The contracts depend on TON continuing to operate as specified. This is outside STRYNER's control and is noted rather than mitigated.

## 3.9.15 Summary

The contract layer's guarantees come predominantly from **absences**: no withdrawal function, no upgrade path, no operator-supplied destination address, no standing authority, no shared balance between challenges, and no dependence on the operator in any recovery path.

Absences are the strongest form of guarantee available, because they cannot be misconfigured, cannot have their keys stolen, and cannot be quietly changed. A capability that does not exist requires no protection.

What the contracts cannot do is make dishonest input honest. That work happens elsewhere, and this chapter should be read alongside 3.7 rather than as a replacement for it. The contracts determine who can move money; verification determines who deserves it. Both are necessary, and neither is sufficient.

# 3.10 Infrastructure Security

### 3.10.1 What the infrastructure is, and what it is trusted with

STRYNER's off-chain infrastructure is deliberately small: an application backend, a database, and the hosting that runs them. Before discussing how it is secured, it is worth restating what it is permitted to do, because the security of the infrastructure layer is defined less by what protects it than by how little it is trusted with.

The infrastructure holds no participant funds, no private keys capable of moving funds, and no authority to alter the outcome of a settled challenge. It records facts — who joined, what steps were reported, how a challenge resolved — and it submits transactions that the smart contracts independently validate. Every consequential action it takes is checked by a system it does not control.

This means a full compromise of the infrastructure, while serious, does not translate into a loss of participant funds. An attacker who took complete control of the backend could corrupt records, disrupt service, and expose the limited data the platform stores, but could not withdraw a participant's deposit, redirect a reward to an address of their choosing, or fabricate a settlement the contracts would honour. The blast radius of an infrastructure breach is bounded by the on-chain design, not by the strength of the infrastructure alone.

### 3.10.2 Hosting and network exposure

The backend runs on managed hosting rather than self-administered servers. This is a deliberate trade: it cedes some low-level control in exchange for a provider whose security, patching, and physical infrastructure are maintained to a standard a small team could not match independently. The database is likewise managed, reached only over an authenticated connection, and never exposed directly to the public internet.

The attack surface reachable from outside is limited to the application's own interface — the set of endpoints the Mini App and Companion legitimately call. There is no administrative interface exposed to the public network, no direct database port open to the internet, and no secondary service running alongside the application that an attacker could reach independently.

### 3.10.3 Secret management

The infrastructure holds two categories of secret that matter: the settlement key that authorises challenge outcomes within fixed limits, and the credentials for the database and third-party services. Neither is stored in the application's source code, and neither is committed to version control.

The settlement key deserves particular note because it is the one secret whose compromise would have on-chain consequences. Its authority is narrow by design — it can sign settlement outcomes but cannot move funds, and it operates only within the limits the smart contracts enforce — but a leaked settlement key would still allow an attacker to submit dishonest outcomes within those

limits. It is therefore held only in the backend's runtime environment, never transmitted, never logged, and rotatable by deploying a new factory should compromise ever be suspected.

### 3.10.4 What a compromise would and would not reach

Stating the failure boundary plainly is more useful than implying it cannot be crossed.

An attacker with full control of the backend could: read the limited personal data the platform stores; corrupt or delete records of steps and participation; disrupt or halt the service; and, with the settlement key, submit dishonest settlement outcomes within the fixed limits the contracts allow.

The same attacker could not: move a participant's deposit; redirect a reward or refund to an address they control, because payouts are reconstructed on-chain from the recipient's own wallet; exceed the economic limits the contracts enforce; or alter a challenge that has already settled and been claimed.

The distinction matters because it defines what participants are actually exposed to. The infrastructure is a point of failure for availability and for the platform's own records. It is not a point of failure for custody of funds, because it was never given custody to lose.

### 3.10.5 Summary

The infrastructure is secured conventionally — managed hosting, no public administrative surface, secrets kept out of code and version control, a settlement key of deliberately narrow authority. But its most important security property is architectural rather than defensive: it holds nothing whose loss would cost a participant their funds. The strongest protection for the infrastructure layer is how little depends on it.

## 3.11 Operational Security

### 3.11.1 Why operational security is treated separately

The preceding sections describe what the system is built to withstand. This one describes the human and procedural layer around it — how the platform is operated day to day, who can change what, and where the realistic risks lie once the code is sound. Operational security is treated as its own concern because the most carefully designed system can still be undermined by how it is run, and because honesty about this layer is more useful to a reader than the impression that the architecture removes all human risk.

STRYNER is, at the time of writing, operated by a single founder. This is stated plainly rather than obscured, because it is the single most important fact about the platform's operational risk profile, and it shapes everything below.

### 3.11.2 The concentration of control, stated honestly

A single operator is both a strength and a weakness, and both should be named.

The strength is that there is no diffuse, poorly-governed team with broad and unaudited access; there is one person, one set of credentials, and a small, comprehensible surface. The weakness is the obvious counterpart: that person is a single point of failure for operational continuity, and the compromise of their credentials is the compromise of the platform's operational control.

The architecture deliberately limits what that control can reach. As established in the preceding sections, operational control does not include custody of participant funds, the ability to move a

deposit, or the power to alter a settled outcome. The concentration of control is therefore a risk to the platform's continuity and to its off-chain records — not to the custody of participant funds, which no operator holds.

### 3.11.3 Change management and deployment

Changes to the running system — contract code, backend logic, the applications — pass through version control before deployment, and the on-chain components carry an additional constraint: the smart contracts governing live challenges cannot be altered. As established in the contract sections, administrative functions shape only future challenges; nothing an operator does reaches a challenge that is already underway. A deployed challenge runs to completion under the rules it was created with, regardless of any subsequent change.

This is a meaningful operational property. It means that the most sensitive component — the one holding value — is the one an operator has the least ability to disturb once it is live. Operational error, or operational compromise, cannot reach into a running challenge and change its terms.

### 3.11.4 Key custody and rotation

The keys that matter operationally are the settlement key, which authorises outcomes within fixed limits, and the administrative key, which deploys and configures future challenges. Neither can move participant funds; both are held outside version control, in environments not exposed to the public network.

Rotation is possible for both. The settlement key is rotatable by deploying a new factory, and the administrative authority is likewise replaceable. The important operational discipline is that the mnemonics backing these keys are stored redundantly enough to survive the loss of any single location, but never so broadly that the number of copies becomes its own exposure. This balance — enough redundancy to recover, little enough to stay contained — is the core of the platform's key-custody posture.

### 3.11.5 Monitoring and the limits of a small operation

The platform observes its own on-chain activity and the health of its off-chain services, and the chain itself provides an independent, publicly verifiable record of every settlement and claim that no operator can quietly rewrite. A participant, an auditor, or an observer can confirm what happened on-chain without trusting the operator's account of it.

It would be dishonest to claim the monitoring maturity of a large organisation. A single-operator platform does not run a staffed incident-response rotation, and this is a genuine limitation. What partially offsets it is architectural: because the consequential state lives on a public chain and the operator cannot alter settled outcomes, the cost of delayed detection is bounded. An operational incident can disrupt service or corrupt off-chain records; it cannot silently drain funds while going unnoticed, because the funds are not in a place the operator can silently reach.

### 3.11.6 Summary

STRYNER's operational security rests on a small, comprehensible surface and an architecture that deliberately limits what operation can touch. The concentration of control in a single operator is named rather than hidden, because it is real; its consequences are bounded by the same design that bounds every other failure in this document — the operator holds no custody of funds and cannot

alter a settled challenge. The honest summary is that operational compromise threatens continuity and records, not participant deposits.

# 3.12 Residual Risk

### 3.12.1 Purpose of this section

The preceding sections each address a specific layer of the system. This one steps back and consolidates: it names the risks that remain after all the protections described above are in place, and points to where each is addressed rather than restating it. A security posture is only honest if it acknowledges what it does not eliminate, and this section exists to make those residual risks explicit in one place.

Nothing here is a newly disclosed weakness. Every item is a risk already discussed in context; the value of gathering them is that a reader can see the whole residual surface at once, and judge it as a whole.

### 3.12.2 Residual risk matrix

The following table maps each significant residual risk to the layer that bounds it and the section that discusses it in full.

| Residual risk | Where it is bounded | Discussed in |
|---|---|---|
| Backend compromise exposes stored data and can corrupt records | On-chain design bounds the blast radius; no custody of funds | 3.6, 3.10 |
| Settlement key compromise allows dishonest outcomes within fixed limits | Contract-enforced economic limits; key rotatable via new factory | 3.6, 3.10, 3.11 |
| A participant reports steps from a manipulated source | Automatic-source-only gate; provenance checks; server revalidation | 3.7 |
| A participant loses access to their own wallet | Outside STRYNER's control by design; no key custody offered | 3.8 |
| A participant approves a malicious transaction outside STRYNER | Per-transaction signing; wallet displays destination and amount | 3.8 |
| Smart contract logic error | Fixed test suite; immutable once deployed; limited admin surface | 3.9 |
| Single-operator continuity and credential compromise | Bounded to continuity and records; no fund custody reachable | 3.11 |
| Service disruption or downtime | On-chain state survives; settled outcomes remain claimable | 3.10, 3.11 |
| Dependence on external services (hosting, chain, health platform) | Discussed as external dependencies with stated roles | 3.10 |

### 3.12.3 The risks that cannot be designed away

Three residual risks deserve naming beyond the table, because they are structural rather than incidental — they follow from choices the platform deliberately made, and cannot be removed without giving up the property that motivated the choice.

The first is that a participant is responsible for their own wallet. STRYNER offers no key recovery, because offering it would mean holding keys, which would reintroduce exactly the custodial risk

the design exists to avoid. The cost of non-custody is that a lost key is lost; this is a deliberate trade, not an oversight.

The second is that the platform depends on the honesty of automatic step data at the point of measurement. Verification checks provenance and rejects manipulable sources, but it cannot audit the internal integrity of a sealed, attested health platform beyond what that platform exposes. The design narrows this risk substantially; it does not claim to eliminate it.

The third is the concentration of operation in a single party. This is bounded, as established, to continuity and records rather than funds — but it remains a real operational risk, and it is named here rather than smoothed over.

#### 3.12.4 The shape of the whole

Read together, the residual risks share a pattern. The risks that remain are, almost without exception, risks to availability, to records, or to a participant's own responsibilities — not risks to the custody of participant funds, because custody was designed out of the system rather than defended within it. Where a risk could touch funds, it is bounded by limits the smart contracts enforce independently of any party STRYNER controls.

This is the intended shape of the security design: not a claim that nothing can go wrong, but an arrangement in which the things that can go wrong are, by construction, kept away from the one thing that matters most.

#### 3.12.5 Summary of Part III

Part III has walked the full security surface: the backend and its trust boundary, verification and its chain of custody, the wallet layer and its two-key asymmetry, the smart contracts and their immutability, the infrastructure and its bounded blast radius, operations and their honest limits, and finally the residual risks that remain. The recurring theme is not the strength of any single defence but the arrangement of the whole — a system built so that its most sensitive asset, participant funds, is protected by structure rather than by trust in any one component. What cannot be eliminated is named. What can be bounded is bounded on-chain. That is the security posture in full.

# Part IV — Protocol Lifecycle

### 4.1 Purpose and scope of this part

Part III described what the system protects and how. This part describes what the system does — the full lifecycle of a challenge from creation to closure, stated as a protocol rather than as a product description. Where earlier parts explained intent, this part specifies mechanics: the states a challenge moves through, what triggers each transition, and what is guaranteed at each step.

The intent is that a reader could follow this part and understand precisely what happens to a participant's deposit at every stage — where it sits, what can move it, and under what conditions it returns. Nothing here introduces new economic rules; it makes the rules already stated in Part II concrete by placing them in sequence.

### 4.2 The lifecycle at a glance

A challenge moves through a fixed sequence of states. Each state has defined entry conditions, defined permitted actions, and a defined path out. The states are:

| State | What it means | How it ends |
|---|---|---|
| Open | The challenge exists and accepts participants | Registration window closes |
| Locked | Registration closed; the activity period runs | Activity period ends |
| Settling | Outcomes are being determined and recorded | A valid settlement is accepted |
| Settled | Outcomes are fixed; rewards are claimable | All rewards claimed, or the claim window ends |
| Expired | The challenge ended without a valid settlement | Deposits are refunded in full |
| Closed | All value has left the contract | Terminal — the contract is finalised |

Two paths lead out of a challenge. The intended path runs Open → Locked → Settling → Settled → Closed. The fallback path, Open → Locked → Expired → Closed, exists so that a challenge which cannot settle for any reason still returns every deposit in full rather than trapping funds.

**4.3 Creation**

A challenge is created permissionlessly. The creator specifies the parameters within their control — the entry amount, the capacity, the number of days, the daily step goal, and the registration and activity windows — and the platform's factory supplies everything the creator must not control: the settlement key and version, the network identity, the payout addresses, the economic splits, and the operational reserve.

This division is a protocol guarantee, not a policy. The creator cannot set the splits in their own favour, cannot name themselves as a payout destination for other participants' unearned deposits, and cannot alter the settlement authority. The parameters a creator supplies define the shape of a challenge; they cannot reach the mechanisms that determine where value flows.

At creation, the challenge contract is deployed with the creator's parameters fixed into it. From this point, those terms are immutable. No participant, including the creator, and no administrative action can change the entry amount, the goal, the windows, or the splits of a challenge that exists.

**4.4 Registration (the Open state)**

While a challenge is Open, any eligible participant may join by depositing the entry amount into the challenge contract from their own wallet. Each deposit is a separate, individually approved transaction; there is no pooled custody the platform holds on participants' behalf, and no participant's deposit is commingled with the platform's own funds.

Joining requires, at the protocol level, that the deposit equal the entry amount plus the participant's own share of the operational reserve and the infrastructure fee. These components are established in Part II; the lifecycle point is that they are collected at join time, from the participant, into the contract — the creator does not front them, and the platform does not advance them.

Registration ends when the registration window closes. The window is defined as a length at creation, not an absolute deadline, so the challenge itself computes when Open ends. No external signal is required to close registration; the transition is a property of time and the contract's own state.

### 4.5 The activity period (the Locked state)

When the registration window closes, the challenge enters Locked. No new participants may join; the set of participants is now fixed for the remainder of the challenge. The activity period runs for the number of days the creator specified, and during this period participants pursue the daily step goal.

Locked is not a stored flag but a derived condition: a challenge is Locked when it is past its registration window and still within its activity period. This matters at the protocol level because it means the transition into Locked requires no transaction and no external trigger — it is simply what the passage of time makes true of a challenge that was Open.

During Locked, deposits sit in the challenge contract. Nothing can move them. There is no action, by any party, that withdraws a deposit during the activity period; the funds are committed until the challenge reaches either settlement or expiry.

### 4.6 Settlement (the Settling state)

When the activity period ends, outcomes must be determined: how many days each participant completed, and therefore what each is owed under the Daily-Unlock economics of Part II. This determination happens off-chain, where the step data lives, and is then submitted to the contract as a settlement.

A settlement is a signed statement of outcomes. It is constructed by the backend, signed with the settlement key, and submitted to the contract, which validates it before accepting it. The contract does not take the settlement on trust: it checks the signature against the settlement key fixed at creation, and it enforces that the outcomes conserve value — that what is distributed equals what was deposited, less the defined splits. A settlement that does not balance is rejected.

The settlement is recorded on-chain as a commitment to the full set of outcomes, structured so that each participant's individual entitlement can later be proven against it. The protocol guarantee is that once a valid settlement is accepted, the outcomes are fixed: no party, including the operator who submitted it, can alter what a participant is owed.

A degenerate case is handled explicitly. A challenge with fewer than the minimum number of participants required to settle does not settle at all; it moves instead to Expired, and every deposit is refunded in full. This ensures that a challenge which never became a genuine contest returns funds rather than distributing them.

### 4.7 Claiming (the Settled state)

Once a challenge is Settled, participants who are owed a reward may claim it. A claim is initiated by the participant, from their own wallet, and the contract reconstructs the claimant's entitlement from the address that signed the claim. This is the security property established in Part III: only the wallet that is owed a reward can claim that reward, because the entitlement is bound to the wallet, not to any field an outside party could supply.

The protocol consequence is that claiming is entirely in participants' hands. The platform cannot claim on a participant's behalf, cannot redirect a claim, and cannot withhold one. A participant who is owed a reward holds the sole and sufficient means to obtain it: their own wallet.

Rewards that are genuinely unclaimed when the claim window ends are handled by the closure logic below, distributed to their defined destination rather than left to sit indefinitely. The treasury and platform shares, being computed on-chain from the pool at settlement and paid directly, are not claimed through this participant-facing path.

### 4.8 The fallback path (the Expired state)

If a challenge reaches the end of its activity period without a valid settlement being accepted — for any reason, whether an operational failure, an insufficient participant count, or any other cause — it becomes eligible for expiry. In the Expired state, the economics of completion do not apply; instead, every participant's full deposit is returned.

This path is the system's guarantee against trapped funds. It exists precisely so that no failure of settlement, however it arises, can leave deposits stranded in a contract. A participant's worst case in a challenge that cannot settle is not loss; it is a full refund of what they put in.

Expiry, like the refund it triggers, is initiated on-chain and reconstructs each refund against the depositor's own wallet, carrying the same property as claims: the refund returns to the wallet that deposited, and to no other.

### 4.9 Closure (the Closed state)

A challenge reaches Closed when all value has left the contract — rewards claimed or routed to their destinations, deposits refunded if expired, defined splits paid. Closure finalises the contract: it returns any remaining balance, including the creator's refundable deployment deposit and whatever operational reserve was not consumed, to the creator, and it terminates the contract.

Closure is the point at which the creator's up-front deployment deposit returns. As established in Part II, this deposit is a temporary commitment, not a fee; it is returned in full at closure, less only the negligible gas the contract actually consumed. A challenge that runs its full course costs its creator nothing but that gas.

The protocol guarantee at closure is conservation: every unit deposited into the challenge has, by the time it closes, either returned to a participant, been distributed as a reward, or been paid as a defined split. Nothing is retained beyond what the rules specify, and nothing is left behind in a closed contract.

### 4.10 Summary of Part IV

Part IV has specified the full lifecycle: creation with parameters fixed and immutable, registration collecting deposits from participants directly, a locked activity period during which nothing can move funds, settlement that fixes outcomes under signature and conservation checks, participant-initiated claiming bound to the claimant's own wallet, a fallback expiry path that refunds in full whenever settlement does not occur, and closure that conserves every unit and returns the creator's deposit. The through-line is that at every state, the movement of a participant's funds is either impossible or in that participant's own hands — never in the platform's.

# Part V — Privacy and Compliance

### 5.1 Purpose and scope of this part

This part addresses what the system knows about its participants and how little that is by design. Privacy at STRYNER is not an added feature but a structural consequence of the same principle that governs the rest of the design: hold as little as possible, so that as little as possible can be lost, leaked, or misused. This part states precisely what data the platform touches, why each piece is necessary, and what the platform deliberately never collects.

Compliance is treated alongside privacy because the two are connected: a system that minimises the data it holds also minimises its regulatory surface. The intent throughout is to describe the actual posture honestly rather than to make claims that outrun what the running system does.

### 5.2 Data minimisation as the governing principle

The platform's approach to personal data can be stated in one sentence: it collects the minimum required to run a fitness challenge, and nothing gathered for its own sake. Every piece of data the system holds exists because a specific function cannot work without it. Where a function can work without a piece of data, that data is not collected.

This is a security property as much as a privacy one, and it was established as such in Part III. Data that is never collected cannot be breached, cannot be subpoenaed, cannot be sold, and cannot be misused. The strongest privacy guarantee the platform can offer for a given piece of information is not to hold it at all, and the design reaches for that guarantee wherever a function permits.

### 5.3 What the platform reads, and why

The core function — verifying that a participant met a daily step goal — requires exactly one category of health data: the step count. The platform reads the number of steps and the provenance information needed to confirm those steps came from an automatic, trustworthy source. It reads nothing else from the health platform.

The following are illustrative of what the platform does not read, even though a health platform may hold them: heart rate, location or GPS traces, sleep data, weight or body metrics, workout details beyond step counts, and any medical information. The verification function has no use for these, so the platform does not request access to them, and no code path exists that would read them.

The provenance information — which application recorded the steps, and by what method — is read not to profile the participant but to make verification possible: it is how the system distinguishes an automatically recorded step count from a manually entered one. This is discussed in full as a verification mechanism in Part III; the privacy point is that even this metadata is used solely to accept or reject a step total, and rejected records never leave the participant's device.

### 5.4 What the platform stores

Beyond the step data needed to resolve challenges, the platform stores the minimum required to operate: a participant's identity as established through Telegram, their public wallet address when they participate in an on-chain challenge, and the record of their participation and outcomes. Each of these is necessary to a specific function — knowing who is in a challenge, knowing which wallet to bind an entitlement to, and knowing how a challenge resolved.

The identity the platform relies on is the one Telegram already provides. The platform does not build a parallel account system, does not collect passwords, and does not gather identifying documents. It does not ask for a legal name, a physical address, a date of birth, or any government identifier as a condition of participation.

The wallet address the platform stores is public blockchain data — visible to anyone with a block explorer — and is stored because entitlements must bind to the wallet that will claim them. Storing a public address discloses nothing that was private; it records a pointer to information already public on the chain.

### 5.5 What the platform never collects

Some categories of data are worth naming explicitly as things the platform does not collect at all, because their absence is itself the guarantee:

- No private keys, seed phrases, or wallet credentials, as established in Part III
- No location or GPS data
- No biometric data
- No health data beyond step counts and their provenance
- No government identifiers or identity documents
- No financial information beyond the public on-chain record of STRYNER's own transactions
- No tracking of activity outside the platform

Each of these is absent by design, not by omission. The system was built so that these never enter it, which means the strongest possible statement about their security — that they cannot be breached because they are not held — is available for each.

### 5.6 The compliance posture, stated honestly

STRYNER's compliance approach follows from its data minimisation and its non-custodial design. Because the platform holds no custody of participant funds and collects no identity documents, its regulatory surface is narrower than that of a custodial financial service. This is a consequence of the architecture, not a claim of exemption from any specific obligation.

The platform is positioned as a skill-based fitness challenge rather than a game of chance, a distinction that matters to how it is regulated and that is reflected throughout the economic design in Part II: outcomes depend on completing a physical goal, not on chance, and a participant's own effort is the determinant of their result. Where legal questions about this positioning remain open in a given jurisdiction, they are treated as matters to resolve with qualified local advice rather than assertions to make in a whitepaper.

It would be dishonest to present the compliance posture as settled in every jurisdiction. The honest statement is that the design deliberately minimises the obligations the platform incurs — by holding no custody, collecting no identity documents, and structuring challenges around skill — and that remaining jurisdiction-specific questions are approached through proper legal channels as the platform grows.

### 5.7 Summary of Part V

Part V has described a privacy posture that is structural rather than declarative: the platform reads only step counts and their provenance, stores only the minimum needed to run challenges, and

never collects entire categories of sensitive data — keys, location, biometrics, identity documents. The compliance posture follows from the same minimalism: a narrower regulatory surface earned by holding less and custodying nothing. The recurring principle, consistent with the rest of the document, is that the safest data is the data never collected.

# Part VI — Economics

### 6.1 Purpose and scope of this part

Part II stated the economic rules; this part explains why they take the form they do. The distinction matters: a mechanism can be specified precisely and still be a bad mechanism if it rewards the wrong behaviour. This part examines the Daily-Unlock model as a system of incentives — what it encourages, what it discourages, and why its particular numbers were chosen — rather than restating the arithmetic already given.

The economic design has one overriding goal: to make honest completion the most rewarding strategy, and to make every other strategy — quitting, gaming, half-effort — strictly worse. Everything below follows from that goal.

### 6.2 The core mechanism restated as an incentive

Under Daily-Unlock, a participant reclaims their own deposit in proportion to the days they complete, and the portion they fail to earn flows into a pool distributed to those who complete every day. Stated as an incentive, this has two simultaneous effects: a participant's own money is at stake against their own consistency, and the money they forfeit becomes someone else's reward for the consistency they lacked.

This dual structure is deliberate. A design that only returned a participant's own deposit in proportion to effort would discourage quitting but would not reward excellence — finishing every day would feel the same as the baseline. A design that only paid a pool to finishers, without staking each participant's own deposit, would invite low-effort entry in hope of a lucky pool share. Combining the two means a participant is defending their own money and competing for others' forfeited money at the same time, and both pressures point toward the same behaviour: complete every day.

### 6.3 Why proportional refund, rather than all-or-nothing

A simpler design would forfeit the entire deposit of anyone who missed a single day. This was considered and rejected, because its incentive structure is perverse at the margin.

Under all-or-nothing, a participant who misses one early day has no remaining reason to continue — their deposit is already forfeit, so the rest of the challenge is effort for nothing. The design would actively produce quitters the moment anyone slipped once. Proportional refund avoids this: a participant who misses a day still has every reason to complete the rest, because each further day they complete returns more of their own deposit. The mechanism keeps a stumbling participant engaged rather than writing them off, which is both better for the participant and better for the challenge.

The proportional structure also better matches the platform's positioning as skill-based rather than punitive. A participant is rewarded for what they achieve, in proportion to achieving it — not subjected to a single cliff that turns one missed day into total loss.

### 6.4 The completion pool and why it pays only full completers

The pool of forfeited deposits pays out only to participants who completed every single day. This threshold is deliberate and sits at the heart of the model's incentive to excel.

If the pool were shared among all participants in proportion to days completed, it would simply be a second proportional refund, and the distinction between finishing every day and finishing most days would nearly vanish. By reserving the pool for full completers, the design creates a sharp, meaningful reward for complete consistency: the difference between completing every day and missing even one is the difference between sharing the pool and not. This is what makes finishing every day worth striving for rather than merely preferable.

A fallback protects against the degenerate case where no one completes every day. Rather than let the pool fall to the operator, it goes to those with the most completed days. This guarantees the pool always rewards the best performers present, and it removes any structural incentive for the operator to prefer mass failure — the operator's share is fixed regardless, and never grows when participants do poorly.

### 6.5 The splits, and why the operator's share is small and fixed

The unearned pool is divided among finishers, treasury, referrals, and platform in fixed proportions established in Part II, with the overwhelming majority going to the participants who completed every day. The operator's own share is deliberately small, and — more importantly — fixed.

The fixed, small operator share is an incentive-alignment choice, not merely a pricing one. Because the platform's share does not grow when participants fail, the platform has no structural interest in designing challenges that produce failure. Its revenue comes from activity happening at all — challenges being created and joined — not from participants losing. This keeps the platform's incentive aligned with the participant's: both benefit most when challenges are popular and completion is achievable, not when goals are punishing and forfeitures are large.

The referral share follows the same logic. It rewards bringing new participants into a challenge, drawn from forfeited deposits rather than from any finisher's earned reward, so that growth is incentivised without diluting what completion pays.

### 6.6 What the model discourages

It is worth naming the behaviours the design makes unrewarding, because a mechanism is defined as much by what it discourages as by what it rewards.

Quitting is discouraged because each completed day returns more of the participant's own deposit; there is never a point at which continuing is worthless. Low-effort speculative entry is discouraged because the pool pays only full completers, so joining without intending to finish forfeits both the pool and part of the deposit. Gaming the step count is discouraged not economically but structurally, through the verification design of Part III — but the economics reinforce it, since a rejected day is a lost day and costs the participant part of their own deposit. And designing punishing challenges to harvest forfeitures is discouraged at the operator level, because the operator's share does not grow with failure.

The consistent pattern is that the design leaves no profitable dishonest or low-effort strategy. The most rewarding thing any participant can do is exactly the thing the platform exists to encourage: move every day.

### 6.7 Conservation and the absence of house edge

A defining economic property, established mechanically in Part IV and worth stating economically here, is conservation: every unit deposited into a challenge is either returned to a participant, distributed as a reward, or paid as a defined split. Nothing is created, and nothing vanishes into an unaccounted balance.

This means there is no house edge in the gambling sense. The platform does not profit from a mathematical advantage over participants; it takes a small, fixed, disclosed share of forfeited deposits, and the rest flows between participants according to their performance. A participant's expected outcome is determined by their own effort and the effort of others, not by an edge structured against them. This is both an economic property and a compliance-relevant one, consistent with the skill-based positioning of Part V.

### 6.8 Summary of Part VI

Part VI has explained the economic design as a system of incentives rather than a set of formulas. Proportional refund keeps stumbling participants engaged; a full-completion pool makes complete consistency sharply worth striving for; a small, fixed operator share aligns the platform's interest with participant success rather than failure; and conservation ensures no house edge. Every element points toward a single behaviour — honest, complete, daily effort — and makes every alternative strictly worse. The economics are not a wrapper around the product; they are the mechanism by which the product does what it claims.

# Part VII — Operations

### 7.1 Purpose and scope of this part

This part describes how STRYNER runs as a living system rather than as a design on paper. It covers what happens when components fail, how continuity is maintained, and what the operational realities of a small platform are. Where section 3.11 (Operational Security) addressed operations through the lens of what an attacker could reach, this part addresses them through the lens of what keeps the system working — and what happens when part of it does not.

The intent, consistent with the rest of the document, is to describe the actual operational posture honestly, including its limits, rather than to project the operational maturity of a larger organisation onto a platform that does not yet have it.

### 7.2 The operational model

STRYNER runs on managed infrastructure with a deliberately small set of moving parts: the smart contracts on-chain, the backend and database off-chain, and the two client applications. Each has a defined role, and the boundaries between them are the same boundaries that define the security model — the contracts hold value and enforce rules, the backend records facts and submits transactions, and the applications present the system to participants.

The operational consequence of this separation is that the components fail independently. A problem in one does not cascade into the others in the way a tightly coupled system would, because the couplings are narrow and the most critical component — the contracts — depends on none of the others to enforce what it enforces.

### 7.3 Failure modes and what each costs

It is more useful to enumerate how the system fails than to assert that it does not. Each component has a characteristic failure mode with a bounded, stated cost.

If the backend fails, the platform cannot submit new settlements or record new activity until it recovers, and the applications lose the data they fetch from it. What does not happen is any loss of funds: deposits sit in the contracts, settled outcomes remain claimable directly from the contracts, and a challenge that reaches the end of its activity period without settlement follows the expiry path and refunds in full. A backend outage is a disruption to service, not a threat to custody.

If the hosting or database fails, the effect is the same class of problem — service disruption bounded by the fact that consequential state lives on-chain. The chain does not depend on STRYNER's infrastructure to remain available, so a participant's ability to eventually claim or be refunded survives an infrastructure outage even if the platform's own interface is temporarily unavailable.

If a client application fails or is unavailable, participants lose the convenient interface to the system, but not the system itself. Because entitlements are claimable directly from the contracts by the wallet that owns them, the applications are the ordinary path to the system, not the only conceivable one.

### 7.4 Continuity and the single-operator reality

The most significant operational risk, named plainly in Part III and restated here in its operational context, is that STRYNER is operated by a single person. This bears on continuity directly: the platform's ongoing operation depends on one operator's continued availability, and this is a genuine limitation rather than one the architecture erases.

What the architecture does provide is a floor beneath that risk. Because settled outcomes are fixed on-chain and claimable directly, and because unsettled challenges expire to full refunds, the failure of the operator to continue does not translate into the loss of participant funds. Participants in settled challenges can claim what they are owed; participants in challenges that never settle are refunded. The operator's continued presence is necessary for the platform to keep functioning and growing; it is not necessary for participants to recover the funds already committed to challenges under way.

This is the honest shape of the continuity risk: it threatens the platform's future, not the participants' committed deposits. Both halves of that statement are true and both are stated deliberately.

### 7.5 Recovery posture

The platform's recovery posture rests on the redundancy of what cannot be regenerated and the reproducibility of what can. The keys that carry authority are backed up redundantly enough to survive the loss of any single location, as established in Part III. The code that runs the platform lives in version control and can be redeployed. The consequential state — participation, settlements, outcomes — lives on a public chain that persists independently of the platform's own infrastructure and can be re-read from that chain if the platform's own records are lost.

This last property is worth emphasising. The chain functions as an authoritative, externally-hosted record of everything that matters economically. Should the platform's own database be lost entirely, the record of who joined, how challenges settled, and what remains claimable is not lost with it, because it was written to a chain the platform does not host and cannot unilaterally alter.

### 7.6 The limits, stated without varnish

A single-operator platform on managed infrastructure has real operational limits, and naming them is more useful than implying they are absent. There is no staffed, around-the-clock incident-response capability. Recovery from a serious operational failure depends on one operator's availability. Monitoring is proportionate to a small operation, not to a large one.

What bounds the cost of these limits is the same architecture invoked throughout this document: because funds are not in a place operational failure can reach, the worst realistic operational incident is a disruption to service and to the platform's own records — not a loss of participant deposits. The limits are real; their consequences are bounded by design. Both facts are stated together because either alone would mislead.

### 7.7 Summary of Part VII

Part VII has described operations as they actually are: a small system with independently-failing components, characteristic failure modes each bounded to service disruption rather than fund loss, a single-operator continuity risk that threatens the platform's future but not participants' committed funds, and a recovery posture anchored in redundant keys, version-controlled code, and an authoritative on-chain record the platform does not host. The operational limits of a small platform are named without varnish; their consequences are bounded by the architecture that bounds everything else in this document.

## Part VIII — Appendices

### 8.1 Purpose of the appendices

The appendices collect reference material that supports the main text without belonging in its flow: the terms used throughout, the structure of the system's components, and pointers to where the authoritative technical detail lives. They are written to be consulted rather than read, and they add no new claims — everything here is a reference to something established in the main body.

### Appendix A — Glossary of terms

The following terms are used throughout this document with specific meanings:

- **Challenge** — A single fitness commitment with a fixed entry, goal, duration, and participant set, implemented as its own on-chain contract.
- **Daily-Unlock** — The economic model by which a participant reclaims their deposit in proportion to days completed, with unearned portions flowing to a completion pool. Defined in Part II, explained as an incentive in Part VI.
- **Completion pool** — The pool of forfeited deposits, distributed to participants who completed every day. Defined in Part II.
- **Settlement** — The signed, on-chain statement of a challenge's outcomes, validated by the contract before acceptance. Specified in Part IV.

- **Settlement key** — The key that signs settlement outcomes within fixed limits. It cannot move funds. Discussed in Parts III and VII.
- **Claim** — A participant-initiated action, from their own wallet, that obtains a reward they are owed. The entitlement is bound to the claiming wallet. Specified in Part IV.
- **Expiry** — The fallback path by which a challenge that does not settle refunds every deposit in full. Specified in Part IV.
- **Closure** — The terminal state in which all value has left the contract and the creator's deployment deposit is returned. Specified in Part IV.
- **Operational reserve** — Per-challenge working capital funding that contract's own on-chain operations; never platform revenue. Defined in Part II.
- **Infrastructure fee** — The small, fixed platform fee; platform revenue. Defined in Part II.
- **Factory** — The contract that deploys challenge contracts and fixes into each the parameters a creator must not control. Discussed in Parts IV and, for security, III.
- **Provenance** — The metadata identifying which application recorded a step count and by what method, used to accept or reject steps. Discussed in Parts III and V.

**Appendix B — Component overview**

The system comprises the following components, each with the role and boundary established in the main text:

| Component | Role | Holds value? | Can move participant funds? |
|---|---|---|---|
| Challenge contract | Holds deposits, enforces rules, pays out | Yes | Only to the wallet the entitlement binds to |
| Factory contract | Deploys challenges, fixes uncontrollable parameters | No | No |
| Backend | Records facts, submits settlements | No | No |
| Database | Stores minimal participation and step records | No | No |
| Mini App | Participant interface inside Telegram | No | No |
| Companion (STRYNER Sync) | Reads and reports verified step counts | No | No |
| Participant's wallet | Authorises entries, claims, refunds | Yes (own funds) | Yes (own funds only) |

The table restates the security model's central asymmetry in one view: value lives in the challenge contracts and in participants' own wallets, and no off-chain component can move a participant's funds.

**Appendix C — Where the authoritative detail lives**

This whitepaper describes principles, economics, and the security posture. The precise implementation detail — contract interfaces, message formats, exact parameters — is maintained separately in a confidential protocol specification kept current with the running contracts, and available to auditors under appropriate terms. Where this document and the running system could ever appear to diverge, the running contracts on-chain are authoritative: they are the system, and this document describes them.

### Appendix D — Document conventions

Throughout this document, a recurring rhetorical choice has been to state what the system cannot do rather than to imply completeness of what it can. This is deliberate. A security claim phrased as an absence — "the platform holds no key that can move funds" — is falsifiable and precise in a way that a claim phrased as a strength is not. Readers are encouraged to treat every such statement as an invitation to verify against the on-chain contracts, which are public.

### 8.2 Closing statement

STRYNER is built on a single principle applied consistently across every layer: hold as little as possible, trust as little as possible, and place the things that matter most beyond the reach of any single component's failure. The security posture is not a set of defences bolted onto a product but the shape of the product itself — a system in which participant funds are protected by structure rather than by trust, in which the most sensitive operations are in participants' own hands, and in which what cannot be eliminated is named plainly rather than hidden. This document has described that system in full. The contracts, on a public chain, are its final and authoritative expression.

# Security Design Trade-offs

# Why the obvious alternatives were rejected

### T.1 Purpose of this chapter

Every design decision in this document was a choice among alternatives, and the alternatives were often the more obvious ones. This chapter examines the significant paths not taken — the "why not just…" questions a thoughtful reader will raise — and states plainly why each was rejected. The value of doing this is twofold: it demonstrates that the design is deliberate rather than accidental, and it exposes the reasoning to challenge. A rejected alternative named openly can be argued with; one left unmentioned cannot.

Each trade-off below follows the same shape: the appealing alternative, what it would buy, what it would cost, and why the cost was judged too high.

### T.2 Why not run everything fully on-chain

The appealing alternative is to put the entire system on-chain — including step verification — so that no off-chain component is trusted with anything, and the whole platform is as verifiable as its contracts.

What it would buy is maximal transparency and the elimination of the backend as a trusted component. What it would cost is impossibility: step data originates in a health platform on a physical device, and there is no way to bring that data on-chain without something off-chain reading it first. A "fully on-chain" verification of physical-world activity is a contradiction — the physical measurement must enter the digital record somewhere, and that somewhere is necessarily off-chain.

The design therefore does not pretend to a fully on-chain ideal it cannot reach. Instead it draws the trust boundary precisely: the off-chain component records and reports, but is structurally unable to move funds or alter settled outcomes, so the thing that cannot be put on-chain is also the thing that cannot cost a participant their deposit. The alternative was rejected because it is unachievable for

physical-activity data; the design's response is to bound what the unavoidable off-chain component can do, rather than to claim it away.

**T.3 Why not verify with GPS or location tracking**

The appealing alternative is to confirm real movement by tracking a participant's location — GPS traces proving they actually walked the distance.

What it would buy is a form of movement verification that is harder to fake than a raw step count. What it would cost is a wholesale abandonment of the platform's privacy posture: continuous location data is among the most sensitive information a person generates, revealing where they live, work, worship, and travel. Collecting it would make the platform a repository of exactly the kind of data Part V is built to avoid holding.

The trade was judged clearly against location tracking. The marginal verification benefit does not remotely justify collecting a participant's movements, and the platform's entire security posture rests on holding less sensitive data, not more. Step counts with provenance checking provide sufficient verification for the platform's purpose without turning it into a location surveillance system. The alternative was rejected because its privacy cost is categorically unacceptable for a marginal gain.

**T.4 Why not use AI or machine-learning verification**

The appealing alternative is to apply a machine-learning model to detect fraudulent activity patterns — to "learn" what cheating looks like and flag it.

What it would buy is a plausible-sounding, adaptive fraud filter. What it would cost is determinism and honesty. A machine-learning verifier produces probabilistic judgements that cannot be precisely explained to a participant whose steps it rejects, cannot be verified by an auditor, and can be gamed by adversaries who learn its blind spots. It would replace a rule a participant can understand — steps must come from an automatic, trustworthy source — with an opaque model whose decisions are neither explainable nor reproducible.

The design prefers a verification rule that is simple, deterministic, and stated plainly over one that is sophisticated and opaque. A participant can be told exactly why a record was rejected; an auditor can confirm the rule; an adversary gains nothing from probing it because it is already public. The alternative was rejected because opacity is a cost, not a feature, in a system that aims to be verifiable.

**T.5 Why not use facial recognition or biometric identity**

The appealing alternative is to bind each account to a real person through facial recognition or other biometrics, preventing one person from running many accounts.

What it would buy is strong identity uniqueness. What it would cost is the collection and storage of biometric data — irreplaceable, uniquely sensitive information that cannot be changed if breached, unlike a password or a key. Holding biometrics would make the platform a target for exactly the kind of breach whose consequences are permanent, and would contradict the data-minimisation principle at the center of the design.

The platform accepts a weaker identity guarantee — identity through Telegram, without biometric binding — precisely to avoid holding biometric data. The economic design further reduces the incentive for multi-account abuse: because each account must stake its own real deposit and the

pool rewards genuine completion, running many accounts means staking many deposits, not harvesting a free advantage. The alternative was rejected because biometric data is the most dangerous category to hold, and the design's economics blunt the problem it would solve.

**T.6 Why not track activity continuously**

The appealing alternative is to monitor a participant's activity continuously throughout the day, rather than reading a daily step total, for richer and harder-to-fake data.

What it would buy is finer-grained activity data. What it would cost is both privacy and purpose. Continuous monitoring would mean the platform holds a minute-by-minute record of a person's physical activity — again, exactly the accumulation of sensitive data the design avoids — for no benefit the platform's actual function requires. The function is to confirm a daily goal was met; a daily total, verified for provenance, answers that question. Continuous tracking answers a question the platform is not asking.

The alternative was rejected because it collects far more than the function needs, and every unit of data collected beyond need is a liability without a corresponding benefit. The design reads what it must — a daily total with provenance — and deliberately declines to watch more closely than its purpose requires.

**T.7 Why not build a dedicated STRYNER tracker or hardware**

The appealing alternative is to control the measurement end-to-end with STRYNER's own tracking application or hardware device, rather than relying on a third-party health platform.

What it would buy is control over the point of measurement — the ability to attest data from a device the platform itself designed. What it would cost is a large increase in both the platform's responsibilities and its attack surface: building and securing a tracker means becoming responsible for the security of the measurement device itself, distributing and maintaining it, and asking participants to trust and adopt a bespoke component instead of the health platform already on their phone.

The design instead relies on the established, sandboxed health platform the participant's device already provides, and verifies provenance from it, accepting that this places part of the trust in that platform's integrity. This is named as a residual risk in Part III rather than hidden. The alternative was rejected because building a bespoke tracker would multiply the platform's responsibilities and adoption friction for a control benefit that provenance-checking a trusted platform substantially provides.

**T.8 The pattern in these rejections**

Read together, the rejected alternatives share a common shape. Almost every one would have bought a marginal gain in verification or identity at the cost of holding more sensitive data, adding opacity, or expanding the platform's responsibilities and attack surface. In each case the design chose the less sensitive, more transparent, more bounded option — even when it meant accepting a weaker guarantee — because the platform's security rests on holding less and trusting less, not on collecting more to defend more.

This is the consistent logic of the whole design, surfaced here through the choices it declined. The measure of a security architecture is not only what it builds but what it refuses to build, and this chapter has named the refusals and the reasoning behind them.