TRACK A — Solana Smart Contract Engineer (Rust/Anchor)

Détail exécutable (PDF séparé)

RBK 2.0

Contents

1 Résumé exécutif du track (1 page)

Positionnement. Ce track forme un **Smart Contract Engineer Solana** capable de livrer des programmes **audit-ready**, conscients des contraintes **Account Model / CPI / compute units** et de la réalité **production** (observabilité, incidents, UX transactionnelle, reproductibilité). Le profil de sortie visé est le **Guardian** : un ingénieur qui sait *concevoir, implémenter, tester, documenter, auditer, durcir et opérer* un smart contract Solana.

Promesse mesurable. À la fin du track, l'apprenant est capable de :

- produire un **repo studio-grade** avec **tests automatisés**, **documentation**, **scripts reproductibles**, **CI** et **runbook**,
- présenter un **threat model** (STRIDE simplifié) et un **mini-audit report** (findings classés, correctifs, preuves),
- maîtriser les patterns **PDA / seeds / constraints / CPI** et éviter les anti-patterns de sécurité,
- livrer un **capstone de production** incluant observabilité, critères d'acceptation et performance budget (compute).

Pré-requis.

- Rust niveau intermédiaire (ownership/borrowing, Result, modules, tests).
- Bases blockchain : transactions, signatures, état, events/logs.
- Discipline d'ingénierie : Git, PRs, revue, documentation.

Différenciation pédagogique (non négociable).

- **Spec-first** : aucun lab sans spécification, critères d'acceptation et plan de tests.
- **Security-first** : threat model et checklists à chaque jalon.
- **Reproductibilité** : scripts de build/test, environnements et démos rejouables.
- **Production mindset** : observabilité, runbook, erreurs explicites, UX transaction.

Ce que ce track n'est pas

Ce track **n'est pas** un apprentissage "rapide" centré sur la syntaxe. Il vise la capacité à **livrer** des programmes Solana robustes, testés, documentés et **auditable** — c'est-à-dire utilisables dans un contexte professionnel.

2 Objectifs mesurables et preuves attendues

Règle. Chaque compétence est évaluée par une **preuve vérifiable** (artefact), un **seuil minimal** et un **outil de vérification**.

Domaine	Compétence	Preuve attendue	Outil & seuil minimal
Solana Model	Account model correct (owners, signers, rent/lamports, sérialisation)	Schéma des comptes + tests de cas limites + logs explicites	Anchor/solana test + couverture tests $\geq 70\%$ des branches critiques
Anchor	PDAs/constraints robustes (seeds, bumps, ownership checks)	IDL + tests négatifs + checklist seeds/constraints signée	Anchor tests + revue "security gate" validée
CPI	Composabilité via CPI sans fuite d'autorité	Schéma call-graph + tests CPI + restrictions d'authority	Tests d'intégration + audit checklist "CPI safety"
Sécurité	Threat model (STRIDE) + findings	Document threat model + mini audit report (min 6 findings dont 2 "High")	Template audit + preuves (PoC tests)
Qualité	CI/CD + reproductibilité	Pipeline CI + scripts build/test + README exécutable	CI verte + "fresh clone works" (checklist)
Production	Observabilité + runbook incidents	Dashboard (métriques/alertes) + runbook + scénarios incidents simulés	PRR validée + MTTR cible simulée ≤ 30 min
Performance	Compute-aware + optimisation raisonnable	Benchmarks CU + justification + budget perf	Rapport perf + respect budget CU défini

Table 1: Compétences cibles vs preuves vérifiables

3 Programme (12 semaines : Semaines 9–20) — modules 1→4

Principe. Chaque semaine produit un **artefact portfolio**. Chaque module se termine par un **jalon** avec **criteria gate** (Go/No-Go).

Semaine	Focus	Objectifs	Lab / livrable	DoD(Definition of Done)
S9	Solana modèle (natif)	Transactions, instructions, comptes, signers, sérialisation	Lab A v0 : Message Board (natif)	Tests unit + tests négatifs + README exécutable
S10	Ownership / sécurité de base	Contrôles owner, signers, erreurs explicites, logs	Lab A v1 : Message Board + permissions	Checklist sécu module 1 signée + couverture cible
S11	Vault/Escrow natif	Gestion de dépôts, conditions de release, cas limites	Lab B : Mini-Escrow/Vault natif	Spec + tests e2e + schéma comptes + démo

Suite page suivante

Semaine	Focus	Objectifs	Lab / livrable	DoD(Definition of Done)
S12	Anchor fundamentals	Macros, Accounts constraints, IDL, erreurs	Lab C : Counter+Vault (Anchor)	IDL propre + tests + events + doc "API surface"
S13	Anchor PDAs	Seeds/bump, authorities, constraints avancées	Lab D : Staking minimal (Anchor)	Tests de sécurité (négatifs) + audit checklist
S14	Anchor composabilité	Events, CPI simple, patterns robustes	Livrable module 2 : "Anchor Pack"	Gate module 2 : PR review + IDL + tests + doc
S15	Architectures avancées	CPI orchestrator, séparation programmes, invariants	Lab E : CPI composability challenge	Call graph + tests CPI + threat model
S16	Token standards / extensions	Token-2022 (ou équivalent selon stack), metadata patterns	Lab F : Token extension + policy	ADR + tests + doc d'intégration client
S17	Innovation / intégration	Indexation, UX tx, actions/blinks (si retenu)	Livrable module 3 : "Composable System"	Gate module 3 : PRR partielle + bench CU
S18	Hardening production	Runbook, logs, métriques, alerting, error taxonomy	Capstone v0 : PRR + monitoring	Checklist PRR 60% + observabilité en place
S19	Performance + sécurité	Compute budget, optimisations, tests de charge	Capstone v1 : perf + security gate	Budget CU respecté + findings traités
S20	Release studio	CI, versioning, docs, démo, post-mortem simulé	Capstone final : release	Gate final : CI verte + audit report + demo

Table 2: Semaine → objectifs → lab → livrable → **DoD**

4 Labs détaillés et critères d'acceptation

Règles communes à tous les labs

- Chaque lab commence par une **spec** (fonctionnalités, comptes, invariants, erreurs).
- Les tests incluent **cas nominaux** + **cas négatifs** (attaques triviales, permissions, double-spend, etc.).
- Chaque livrable inclut : **README exécutable**, **scripts**, **schéma des comptes**, **journal des décisions** (ADR).

Lab A — Message Board (natif Solana)

Objectif. Construire une primitive on-chain simple, mais **correcte** : stockage sérialisé, autorisation, erreurs explicites, logs.

Critères d'acceptation (Gate Lab A).

- README "fresh clone" : un clone vierge compile et passe les tests.
- Tests : ≥ 12 tests dont ≥ 5 négatifs.

Fonctionnalité	Entrées / Comptes	Tests attendus (mesurables)
Créer un message	signer auteur + compte message (PDA ou compte dédié)	création OK ; échec si auteur absent ; taille max contrôlée
Modifier un message	signer auteur + compte message	échec si non-auteur ; logs explicites ; état cohérent
Lister via indexation off-chain	events/logs + indexer (simulé)	events émis ; schéma d'event stable ; doc d'intégration
Gestion erreurs	codes d'erreur + messages	tests sur erreurs attendues ; pas d'erreurs silencieuses

Table 3: Spécification Lab A (Message Board)

- Documentation : schéma des comptes + table "Instruction → comptes → erreurs".

Lab B — Mini Escrow / Vault (natif Solana)

Objectif. Implémenter un escrow/vault avec règles de release et **anti-abus**.

Règle	Invariants	Tests attendus
Dépôt	fonds conservés dans vault ; owner vérifié	dépôt OK ; échec si mauvais owner ; double dépôt contrôlé
Release conditionnel	release uniquement si condition satisfaite	release OK ; échec si condition fausse ; reentrancy-like patterns (simulés)
Cancel / timeout	cancel seulement par rôle autorisé / selon délai	cancel OK ; échec si rôle incorrect ; délais testés
Journalisation	events pour opérations critiques	events présents ; payload stable ; doc d'event

Table 4: Invariants et tests Lab B (Escrow/Vault)

Lab C — Counter+Vault (Anchor fundamentals)

Objectif. Maîtriser IDL, macros, constraints, events, error codes.

Instruction	Comptes (résumé)	Erreurs & tests
initialize	authority signer + state PDA	échec si déjà init ; seeds correctes
increment	authority signer + state PDA	échec si non-authority ; event émis
deposit/withdraw	user signer + vault + state	échec si fonds insuffisants ; logs ; invariants

Table 5: IDL et surface API Lab C

Lab D — Staking minimal (Anchor PDAs & constraints)

Objectif. PDAs déterministes + constraints robustes + tests négatifs.

Critères d'acceptation.

- Tous les accès sensibles protégé par checks explicites (authority/owner/signers).
- Seeds documentées ; tests sur seeds ; "bump mismatch" géré.
- Tests de retrait anticipé/illégitime → échec explicite.

Lab E — CPI composability challenge (Architecture avancée)

Objectif. Construire un orchestrateur qui appelle un programme "service" via CPI en conservant un modèle d'autorité sain.

Exigences CPI (non négociables)

- Aucun CPI ne doit introduire une autorité implicite ou réutilisable par un tiers.
- Les comptes en CPI doivent être minimaux et vérifiés (owner, seeds, signers).
- Le call-graph est documenté et testé.

Lab F — Token extension / policy (Token-2022 ou équivalent)

Objectif. Démontrer la capacité à gérer des métadonnées/policies et leur validation (off-chain + on-chain checks).

Livrables.

- ADR : choix d'extension/pattern.
- Tests : policy enforcement (positifs + négatifs).
- Guide intégration client (front) : "how to verify".

5 Rubrique de notation (standard audit) — total 100

Axe	Poids	Critères (exemples mesurables)
Sécurité & threat model	25	threat model complet ; findings classés ; corrections justifiées ; tests "négatifs"
Tests & qualité	20	tests unit+int ; cas limites ; CI stable ; couverture sur chemins critiques
Architecture & invariants	15	invariants explicités ; schémas comptes ; call-graph CPI ; décisions ADR
Production & observabilité	15	runbook ; logs ; métriques ; alerting ; scénarios incidents simulés
Performance (compute)	10	budget CU ; benchmarks ; choix d'optimisation argumentés
Docs & demo	15	README exécutable ; doc API ; diagrammes ; démo reproductible

Table 6: Rubrique d'évaluation Track A (100 points)

Condition d'échec immédiate (Fail conditions)

- absence de tests négatifs sur les chemins d'autorité,
- absence de documentation des seeds/PDA sur un programme Anchor,
- livrable non reproductible (fresh clone ne compile pas / tests KO),
- absence de threat model sur capstone.

6 Stack outillage et standards repo

Catégorie	**Outils**	**Standard attendu (artefact)**
Toolchain	Rust toolchain, Solana CLI, Anchor	scripts build/test ; versions documentées ; "fresh clone"
Testing	anchor test, tests TS, tests Rust	unit + integration + négatifs ; rapports CI
Qualité	fmt/clippy, lint, pre-commit	format stable ; lint clean ; conventions repo
Observabilité	logs, métriques, dashboards (selon stack)	runbook + alert rules + tableau "signaux"
Docs	README, schémas, ADR, threat model	dossier /docs ; templates remplis ; lien demo

Table 7: Stack Track A — outillage et standards

Standards repo (minimum)

- `/programs` (code on-chain), `/tests`, `/scripts`, `/docs`
- `README.md` exécutable + prérequis + "how to run tests"
- `docs/architecture.md`, `docs/threat-model.md`, `docs/adr/`
- CI : lint + tests + artefacts (rapport) ; badge de status

7 Portfolio minimal et employability pack

Portfolio minimal (obligatoire).

- **Repo 1 — Solana Native Primitive** : Lab A/B finalisé, tests négatifs, doc comptes.
- **Repo 2 — Anchor Program** : IDL propre, PDAs/constraints robustes, guide intégration client.
- **Repo 3 — Capstone Studio** : hardening + observabilité + perf budget + audit report.
- **1 mini audit report** : findings, sévérité, correctifs, preuves tests.
- **1 runbook** : incidents types + actions + métriques/alerting.

Employability pack (recruteur-ready).

- 1 page "*What I shipped*" (liens repos + démos).
- 1 page "*Security posture*" (threat model + checklists + exemples corrections).
- 1 page "*Performance posture*" (compute budgets, benches, arbitrages).

8 Annexes du track (templates & checklists)

A) Security checklist Solana/Anchor (réutilisable)

Contrôle	Comment vérifier (preuve)
Owner checks	asserts explicites + tests négatifs
Signer checks	requires signer + tests “missing signer”
PDA seeds/bump	seeds documentées + tests seeds + collisions évitées
Account constraints (Anchor)	constraints justifiées + tests violations
CPI safety	call-graph + accounts minimal + authority non escaladable
Error taxonomy	erreurs explicites + mapping doc + logs
Replay / double spend logique	invariants + tests de répétition
Overflow / underflow (logique)	checks + tests limites
State machine cohérente	diagramme + tests transitions invalides

Table 8: Security checklist Solana/Anchor

B) Template ADR (Architecture Decision Record)

ADR Template (copier-coller)

Titre : ADR-## — *[Décision]*
Contexte : *[Problème, contraintes]*
Options : *[Option A / B / C]*
Décision : *[Choix]*
Rationale : *[Pourquoi]*
Conséquences : *[Trade-offs]*
Risques : *[Risques]*
Mitigations : *[Contrôles/tests]*

C) Template Threat Model (STRIDE simplifié)

Threat Model Template (STRIDE simplifié)

Système : *[Nom + périmètre]*
Actifs critiques : *[fonds, autorisations, état]*
Hypothèses : *[RPC, indexer, user behavior]*
Surfaces d’attaque : *[instructions, CPI, accounts]*
Menaces (STRIDE) :
- Spoofing : *[...]*
- Tampering : *[...]*
- Repudiation : *[...]*
- Information disclosure : *[...]*
- Denial of service : *[...]*
- Elevation of privilege : *[...]*

Contrôles : *[checks, constraints, tests]*
Tests associés : *[négatifs, invariants]*

D) PRR — Production Readiness Review (tableau)

Domaine	Critère	Preuve
Qualité	CI verte + tests + lint clean	logs CI + badge + rapports
Sécurité	threat model + checklist signée	docs + PR review
Docs	README exécutable + API surface	/docs + guide
Observabilité	logs + métriques + alertes	dashboard + runbook
Perf	budget CU + bench	rapport bench
Release	versioning + changelog	tag + notes

Table 9: PRR (Production Readiness Review) — Track A

9 Figures indispensables (TikZ)

Figure 1 — Account model / instruction flow

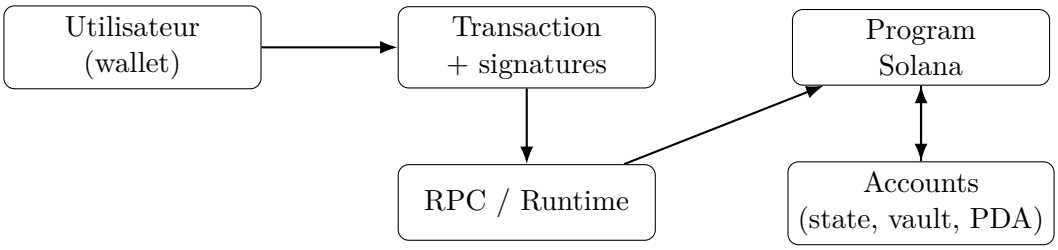

Figure 1: Account model / instruction flow (vue simplifiée)

Figure 2 — CPI call graph (simplifié)

Règle : comptes minimaux, owner checks,
authority explicitement contrôlée.

Figure 2: CPI call graph (orchestrateur → service → programme système)