Certus

Verifiable Compute Without Trust Assumptions

Evan W.

evan@certuscompute.com

@CertusCompute

November 2025

Version 1.0

Contents

1 The Problem Nobody Has Solved

Blockchains are great at one thing: verifying that money moved. Alice sends 10 ETH to Bob. Every validator re-executes the transaction, checks the signature, updates the balances. Consensus. Done.

But what if Alice wants to pay Bob to render a video? Or run a machine learning model? Or process a gigabyte of scientific data?

You can't re-execute that on-chain. A 10-second video transcode costs $0.01 on AWS. On Ethereum, it would cost $10,000 in gas and still fail because of block gas limits. This isn't a scaling problem, it is a fundamental architectural mismatch. Blockchains achieve consensus through redundant execution. Compute is expensive. The math doesn't work.

So how do you verify that Bob actually did the work correctly?

1.1 The Oracle Approach

Get a committee of validators to each do the work off-chain and vote on the result. If 7 out of 10 nodes say "output is X," that becomes truth. This works for simple data (price feeds, weather), but breaks down for complex computation. What if the nodes disagree? What if 6 collude? What if the computation has a bug? There's no ground truth, just a popularity contest.

1.2 The TEE Approach

Run the computation inside a trusted execution environment (Intel SGX, AWS Nitro). The hardware signs an attestation proving it ran specific code on specific inputs. This is fast and elegant until someone finds a vulnerability. Intel has issued 200+ security patches for SGX. One break compromises every computation ever attested. You're trusting hardware manufacturers, which means you're trusting nation-states that can compel hardware manufacturers.

1.3 The ZK Approach

Generate a cryptographic proof that computation was done correctly. This is the gold standard: instant verification, zero trust assumptions, perfect security. It's also completely impractical for general compute. Proving time is 100–1000× the execution time. A 10-second video transcode takes 20 minutes to prove. A 1-hour ML training run takes 6 weeks to prove. The proof itself costs $50–$500 to generate. Nobody is going to pay that for commodity compute.

1.4 The Optimistic Rollup Approach

Execute off-chain, post results on-chain, allow challenges. This is closest to what we need, but it's designed for verifying blockchain state transitions (complex, stateful, non-deterministic) not for verifying arbitrary computation. Fraud proofs require interactive bisection over execution traces. Challenge periods are 7 days because state is complex and disputes are expensive. Capital is locked. Only sophisticated operators run fraud provers.

The Current State: Oracles require trust. TEEs require hardware trust. ZK proofs are too slow and expensive. Optimistic rollups are too slow and complex.
Nobody has built trustless, general-purpose, economically viable verifiable compute.
Until now.

2 The Insight: Determinism Makes Fraud Provable

The breakthrough isn't a new cryptographic primitive or a faster proof system. It's recognizing that **if computation is deterministic, fraud becomes binary and provable with minimal on-chain work.**

Here's the key: When two parties disagree about a deterministic computation, exactly one of them is lying. There's no ambiguity. Same inputs always produce same outputs. If Alice claims `SHA256("hello") = 0xabc123...` and Bob claims `SHA256("hello") = 0xdef456...`, one of them is wrong. You can verify this by running `SHA256("hello")` once. Not voting. Not consensus. Just execution.

Now extend this:

- If computation is deterministic, fraud is binary (true/false, no gray area)
- If fraud is binary, you don't need M-of-N consensus (one honest verifier is enough)
- If one honest verifier is enough, collusion is exponentially harder
- If you can narrow the dispute to a tiny segment, on-chain verification is cheap
- If on-chain verification is cheap, challenge periods can be short
- If challenge periods are short, capital efficiency is high

This is the entire design.

3 How It Actually Works

3.1 Deterministic Wasm Execution

We use WebAssembly as the execution environment. Not because it's trendy, but because it's the only widely-adopted format that's deterministic by default.

Wasm has no threads, no syscalls, no undefined behavior, no access to the filesystem or network. It's a pure computation sandbox. You give it bytes in, it gives you bytes out. The same bytes in always produce the same bytes out, on any machine, at any time, forever.

This is stronger than it sounds. Most "deterministic" systems have edge cases:

- EVM has `block.timestamp` and `block.difficulty` (non-deterministic across validators)
- RISC-V has memory consistency issues across implementations
- Native x86 has undefined behavior in corner cases
- Even LLVM IR has undef values that can vary across compiler versions

Wasm doesn't. The spec is tight. The semantics are clear. We validate modules before registration to ensure they don't use floating-point operations (non-deterministic rounding) or host imports (external calls). What you're left with is pure, deterministic computation.

FFmpeg transcodes to Wasm. TensorFlow Lite compiles to Wasm. SHA-256 hash libraries, image processing, scientific simulation—anything that doesn't need I/O compiles to Wasm and runs deterministically.

3.2 The Execution Flow

3.2.1 1. Client Publishes a Job

Alice wants a video transcoded. She:

- Submits the Wasm module hash (FFmpeg compiled to Wasm)
- Submits the input data hash (raw video file)
- Escrows payment ($50 USDC) + bond ($5 to prevent griefing)
- Specifies collateral ratio (2× = executor must deposit $100)

The job gets a unique ID derived from the content:

```
SHA256(wasmHash || inputHash || clientPubKey || nonce)
```

This prevents replay attacks and makes every job globally unique.

3.2.2 2. Executor Accepts and Runs

Bob sees the job. He downloads the Wasm module and input data (from IPFS, Arweave, wherever; doesn't matter as hashes verify authenticity). He runs the job in a deterministic sandbox with fuel limits (instruction budget) and memory limits (bytes). He computes `outputHash = SHA256(output)`.

Bob now posts his receipt on-chain:

- Job ID
- Output hash (32 bytes)
- Ed25519 signature

His $100 collateral is locked. If he lied, he loses it all.

3.2.3 3. Verifiers Check

Three verifiers are randomly selected (weighted by staked CERTUS tokens). They each independently download the Wasm and input, run the computation, and check if their output hash matches Bob's claim.

If everyone agrees: Alice finalizes, Bob gets paid, verifiers get their fee.

If anyone disagrees: They challenge, and we do bisection.

3.3 Bisection: The Fraud Proof Protocol

This is where it gets interesting. When a verifier (let's call her Carol) detects fraud, she doesn't need to re-run the entire computation on-chain. That would be prohibitively expensive. Instead, we use interactive bisection to narrow down the dispute to a tiny segment, then verify just that segment on-chain.

3.3.1 How Bisection Works

Imagine the computation produces 1,000,000 bytes of output. Bob claims the output hash is H_{Bob}. Carol claims it's H_{Carol}. They disagree.

Round 1: Carol says "your output is wrong." Bob responds with the hash at the midpoint (500,000 bytes): H_{mid}.

Round 2: Carol recomputes the first 500k bytes. She either agrees with H_{mid} (dispute is in second half) or disagrees (dispute is in first half). She picks one half. Bob responds with the hash at the new midpoint.

Rounds 3–20: Continue bisecting.

After ~20 rounds ($\log_2$ of 1M), the disputed range is narrowed to 32 bytes. Now we execute those 32 bytes on-chain using Arbitrum Stylus (native Wasm execution in the EVM environment). The on-chain result is ground truth. If it doesn't match Bob's claim, Bob is slashed. His $200 (100% of 2× collateral) goes to: 80% to Alice (client refund), 20% to Carol as fraud bounty.

Gas cost: Bisection requires ~3.8M gas total. At 0.1 gwei and $3000 ETH, that's about $0.38. Compare to naive re-execution (125M gas = $12.50). We achieve 97% gas savings by bisecting.

3.3.2 Why This Works

Bob can't cheat bisection. At each round, he must provide a hash for the midpoint. If he lies, Carol catches him in the next round. If he provides the correct hash, the dispute moves to one half. Eventually we reach a 32-byte segment that's cheap to verify on-chain. Bob can't hide. Determinism guarantees Carol can reproduce the exact same output.

3.4 The Economic Layer

The fraud proof protocol only works if the economics make lying unprofitable. Here's how:

Collateral: Bob deposits 2× the payment ($100 collateral for $50 job). If he's honest, he gets it back plus payment. If he lies, he loses it all. The collateral ratio is fixed at 2× for all executors, ensuring fair competition and making fraud economics simple.

Fraud bounty: If Bob lies, Carol earns 20% of the slashed collateral ($40 on a $100 job with 2× collateral). It costs her $0.38 in gas to prove fraud. She nets $39.62. This is highly profitable, so she will always challenge if she detects fraud.

Verification fee: Fees are tiered by job value (1–2% with minimums from $0.10 to $150). This makes small jobs economical while extracting value from large jobs. At scale (500 jobs/month per verifier averaging $10 fee), verifiers earn $150+/month from fees alone.

Expected value for Bob:

- Honest: +$50 per job (100% certain)
- Fraud: -$200 per job (100% certain, because computation is deterministic and collateral is 2×)

There's no probabilistic detection here. Carol doesn't "maybe" catch fraud. She **definitely** catches it, because she gets a different output hash when she re-runs the deterministic computation. Fraud is suicide.

4 Why This Is Hard (And How We Solved It)

Building this isn't just a matter of combining existing pieces. There are real technical challenges.

4.1 Challenge 1: Determinism Is Fragile

Most software isn't deterministic. Hash maps iterate in random order. Timestamps depend on system clocks. Floating point operations round differently on different CPUs. Multi-threading introduces race conditions. Memory allocators use ASLR.

Our solution: Enforce determinism at the Wasm module level. We scan modules before registration and reject any that use:

- Floating-point operations (use fixed-point or integer math only)
- Non-deterministic instructions (random, time, threading)
- Host imports (environment access)
- Excessive memory (limit to prevent DOS)

We provide a deterministic stdlib that developers can use. For operations that need randomness (like Monte Carlo simulation), you provide the random seed as input. For operations that need time (like timestamping), you provide the timestamp as input. Everything is explicit.

This is actually a feature, not a limitation. Deterministic code is testable code. If you can't reproduce the output from the inputs, you can't test it properly anyway.

4.2 Challenge 2: State Explosion in Bisection

Naive bisection would require Bob to store the entire execution trace (intermediate hashes at every step). For a 1GB output, storing hashes at exponential intervals still requires megabytes of data.

Our solution: Bob doesn't store intermediate hashes. He recomputes them on-demand during bisection. When Carol challenges round 5 and asks for the hash at byte position 781,250, Bob re-runs the computation up to that point and provides the hash. This is fast (execution is near-native speed in Wasm) and requires zero storage.

The challenge period is 5 minutes per round for bisection responses. If Bob doesn't respond in time, he's automatically slashed. But here's the key: the initial challenge window is **1 hour**, not 5 minutes. This gives verifiers in all timezones time to wake up, recompute, and submit fraud proofs. We learned this the hard way: 5-minute windows enabled timezone arbitrage attacks. An hour kills that vector completely.

4.3 Challenge 3: Verifier Selection Randomness

If Bob can predict which verifiers will be selected, he can bribe them or run Sybil verifiers. We need unpredictable selection that happens after Bob commits to his result.

But more importantly: we need to make Sybil attacks and collusion attacks economically impossible, not just hard to predict.

Our solution: We use 11 entropy sources to seed the random selection:

```
block.timestamp, block.prevrandao, block.difficulty, block.number,
jobId, wasmHash, tx.origin, msg.sender,
blockhash(block.number-1), blockhash(block.number-2),
blockhash(block.number-3)
```

This combines miner-controlled randomness (prevrandao), client-controlled data (jobId, tx.origin), and historical block data (blockhashes). Manipulating this would require an Arbitrum sequencer to frontrun the submission, control three historical block hashes, and predict the job ID. The attack cost exceeds the potential gain by orders of magnitude.

Additionally, selection happens after Bob submits his receipt. He can't target specific verifiers because he doesn't know who will be selected. The protocol also implements MEV protection through a commit-reveal scheme: verifiers commit fraud proof hashes first, then reveal after a 2-minute delay, preventing frontrunning.

4.4 Challenge 4: Data Availability

If Alice or Bob withholds data (Wasm module or input), verification is impossible. We can't force them to publish data on-chain (too expensive for large inputs).

Our solution: Two-tier data availability that's actually enforced, not hoped for:

Inputs ≤100KB: Stored directly on-chain in calldata. Costs ~$2 on Arbitrum, but the data is permanently available. Indexers cache it. Verifiers can always access it. No IPFS unpinning bullshit, no "data disappeared" excuses. It's there forever.

Inputs >100KB: Must use Arweave (permanent storage, $0.01/MB) with a 10% bond. If the data becomes unavailable during the challenge window, the bond is slashed and the job is cancelled. Alice loses her bond. Bob gets refunded.

This creates skin-in-the-game. Alice can't submit a job, wait for Bob to execute, then unpin the data to avoid a challenge. She'd lose 10% of the payment. Bob can't claim "I can't defend myself, data is gone" because the data's availability is enforced by economic penalty.

For the MVP, Wasm modules (≤24KB) are always stored on-chain. The contract validates module determinism before registration, rejecting any that use floating-point operations, WASI imports, or non-deterministic instructions. Future versions might support larger modules via blob storage, but for general compute, 24KB is plenty (FFmpeg transcoder, ML inference engine, image processor all fit within this limit).

4.5 Challenge 5: Capital Efficiency vs Security

High collateral ratios (3×) make fraud expensive but lock up executor capital. Low ratios (1×) free up capital but make fraud cheaper. How do we balance this?

Our solution: Fixed 2.0× collateral for all executors. No dynamic reputation, no tiered rates, no oligopoly formation. Every executor competes on equal footing. New entrants can compete immediately without capital disadvantage.

The economics are simple: fraud costs you $200 (100% collateral slash) to attempt stealing $100 (job payment). Even with a 25% chance of detection, the expected value is negative. But detection isn't 25% probabilistic. It's deterministic and certain. Three independent verifiers each re-run the job. All three must collude or be offline for fraud to succeed. This doesn't happen.

We add harsh penalties for repeat offenders: First fraud attempt gets you a 30-day ban. Second fraud attempt gets you permanently banned. Your address is marked on-chain. Clients can see this and blacklist you. The economic and reputational damage is severe enough that rational executors never attempt fraud.

This creates fair competition while maintaining security:

- All executors have same capital requirements (no first-mover advantage)
- Fraud is always -EV (deterministic detection × $200 loss ≫ $100 gain)
- No complex reputation system to game or manipulate

5 The Token Model (And Why It's Not Bullshit)

Most crypto projects have a token because they need to fundraise. The token has no real utility, just governance and staking rewards that inflate the supply. This is backwards.

CERTUS exists because the protocol needs scarce resources for security and efficiency. The token is a coordination mechanism, not a fundraising tool.

5.1 Seven Utility Mechanisms

1. Verifier selection boost: Stake 10k/50k/200k CERTUS → 1.2×/1.5×/2.0× higher selection probability → proportionally more jobs → proportionally more income. Tiered structure prevents whale dominance while rewarding commitment. Maximum 2× boost ensures decentralization.

2. Fee discount: Pay 50%/100% of fees in CERTUS → receive 25%/40% discount. Partial payment option allows gradual adoption. This reduces sell pressure on CERTUS (clients buy and spend immediately) while giving clients real savings. High-volume clients save thousands per month.

3. Revenue share: Lock CERTUS → earn veCERTUS → receive 50% of protocol fees in USDC. This is real yield, not inflation. At 500k jobs/month, veCERTUS holders earn 30% APY in stablecoins from actual protocol revenue.

4. Executor insurance pool: Stake 50k CERTUS → join insurance pool that covers 50% of slashing losses (funded by 10% of protocol fees). This reduces executor risk without creating capital efficiency advantages that lead to oligopolies.

5. Priority execution: Burn 100 CERTUS → job gets priority matching. This is deflationary and serves real use cases (time-sensitive jobs like real-time inference).

6. Governance: 1 CERTUS = 1 vote on fee parameters, treasury allocations, upgrades. Votes have real economic impact (changing fees from 1% to 2% doubles protocol revenue). No theater.

7. Buyback & burn: 30% of protocol fees → market buy CERTUS → burn. At scale, this removes 9M tokens/year (9% of supply). Token becomes deflationary as protocol grows.

5.2 Why This Isn't Vaporware

The token derives value from protocol usage, not speculation. Revenue comes from clients paying for compute (real dollars), not from token inflation. The yield is paid in USDC from fees, not from minting new tokens.

At 500,000 jobs/month ($25M payment volume), the protocol generates $3M/year in fees. This is real revenue. It funds:

- $1.5M/year to veCERTUS stakers (paid in USDC)
- $900k/year to buyback & burn (removes supply)
- $600k/year to treasury (development, audits, operations)

The token price floor is set by yield. If veCERTUS earns 30% APY in USDC, rational actors will buy and stake until yield falls to market rate (15–20%). This puts a floor around $0.50–$1.00 per token at scale.

5.3 The Bootstrap Problem

Early on, there aren't enough jobs to fund verifier income. We solve this with dynamic treasury subsidies that guarantee minimum income during the bootstrap phase:

- Bootstrap phase (months 0-6): $100/month minimum per verifier
- Growth phase (months 7-24): Declining subsidies from 100% to 10%

- Mature phase (24+ months): Market-driven income only
- Subsidy formula dynamically adjusts based on actual fee income and emissions

Total cost: ~$180k for base-case growth over 48 months. Treasury holds 30M CERTUS ($3M at $0.10). Bootstrap cost is 6% of treasury, sustainable with 94% remaining for insurance pool, grants, and emergency reserves.

At 5 jobs/month (worst case cold start), a verifier earns $100 dynamic subsidy + $0.02 fees + $50 in CERTUS emissions = $150/month on a $1,000 stake. That's 1,800% APY. Income is guaranteed through dynamic subsidies that adjust to market conditions.

By month 12, the system starts approaching self-sustainability. By month 36, subsidies drop to near-zero and verifiers earn $150–200/month from fees + emissions alone.

6 What This Enables (And What It Doesn't)

6.1 What You Can Build

Anything that's deterministic and doesn't need I/O. Specifically:

- Video/audio transcoding (FFmpeg, libav)
- Image processing (filters, compression, format conversion)
- Cryptographic operations (hashing, signature verification, key derivation)
- Data transformations (CSV to Parquet, JSON to protobuf)
- Scientific simulation (Monte Carlo, physics engines, molecular dynamics)
- ML inference (run trained models on new data)

Real examples:

- **Verifiable NFT generation:** Prove that generative art follows the claimed algorithm
- **Trustless trading bots:** Prove that your bot executes the strategy you published
- **Fair lotteries:** Prove that winner selection used claimed RNG with blockchain entropy
- **Provable ML predictions:** Prove that a model produced specific outputs on specific inputs

6.2 What You Can't Build

- Anything that needs filesystem access (databases, file I/O)
- Anything that needs network access (API calls, web scraping)
- Anything that needs true randomness (we require explicit random seeds)
- Anything non-deterministic (e.g., multi-threaded programs with race conditions)

These aren't limitations of Certus, they are fundamental incompatibilities with verifiable compute. If your program gives different outputs on the same inputs, verification is impossible.

6.3 What About Training ML Models?

Training is iterative and often non-deterministic (random weight initialization, data shuffling, dropout). We can't verify training directly. But we can verify:

- Inference (running a trained model on new data)
- Deterministic training (fixed seed, fixed data order, no dropout)

Most valuable ML use cases are inference, not training. Verifying that GPT-4 gave you a specific response to a specific prompt is valuable. Verifying that GPT-4 was trained correctly is less relevant.

6.4 What About Interactive Programs?

Programs that take user input during execution can't be verified with our model. The entire input must be known upfront. This rules out:

- Interactive games (user makes moves during play)
- Chatbots (user provides prompts during conversation)
- Real-time simulations (parameters change during execution)

You can work around this by batching: Collect all inputs upfront, run the program with all inputs at once, verify the output. Not ideal, but it works for many use cases.

7 Why Hasn't Someone Done This Already?

They have, sort of. Arbitrum and Optimism use fraud proofs with bisection. Truebit proposed verifiable computation in 2017. Golem has been trying to build decentralized compute since 2016.

So what's different?

7.1 Arbitrum/Optimism

Designed for verifying blockchain state transitions, not arbitrary compute. Their fraud proofs work on EVM execution traces, which are complex (thousands of opcodes, global state, non-deterministic block data). Challenge periods are 7 days because state proofs are expensive. They're optimized for throughput and finality, not for compute workloads.

We're doing the opposite: stateless Wasm execution (pure functions), simple fraud proofs (just output hashes), short challenge periods (5 minutes), optimized for compute jobs.

7.2 Truebit

Brilliant design, ahead of its time. They proposed the verification game (analogous to our bisection protocol) in 2017. The issue was market fit. Ethereum gas was cheap in 2017, so verifiable compute didn't have a clear use case. They also didn't solve the incentive problem as verifiers had to do work speculatively, hoping to catch fraud. With rare fraud, verifier income was insufficient.

We solve this with deterministic execution (fraud is certain, not probabilistic) and verification fees (verifiers earn from every job, not just fraud). Verifiers don't speculate—they're paid to check every result.

7.3 Golem

Focused on marketplace and reputation, not verification. They assume executors are mostly honest and use reputation to weed out bad actors. This works for non-critical workloads but doesn't give you security guarantees. One bad executor can steal from naive clients.

We use cryptographic verification, not reputation. An executor with 10,000 successful jobs can still be proven wrong on job 10,001. No trust needed.

7.4 Why Now?

Three things have changed:

1. Arbitrum Stylus gives us native Wasm execution in the EVM. Before this, we'd need a custom VM or off-chain proof verification. Stylus makes on-chain Wasm verification cheap and practical.

2. Wasm maturity: FFmpeg, TensorFlow, LLVM backends—everything compiles to Wasm now. In 2017, Wasm was a browser thing. In 2025, it's the universal compilation target.

3. Economic scale: At $3000 ETH and 0.1 gwei gas, fraud proofs cost $0.38. At $300 ETH (2017 prices), they'd cost $3.80—too expensive relative to job values. Gas has gotten cheaper, ETH has gotten more valuable, and L2s have made everything 10–100× cheaper. The economics now work.

8 The Threats (And How We Mitigate Them)

8.1 Executor Lies, Nobody Challenges

Can't happen. Computation is deterministic. If Bob submits outputHash = H_{Bob} and the correct hash is H_{Carol}, any verifier who re-runs the computation will see the mismatch. Carol earns $40 (20% of 2× collateral) for catching fraud at a cost of $0.38. She will always challenge.

8.2 Executor Bribes All Selected Verifiers

Selection happens after Bob submits, and he needs to bribe all 3 primary verifiers (Bob doesn't know who the 3 backups are either). Bribe must exceed fraud bounty ($10+ per verifier). But we have multiple layers of defense:

1. Geographic diversity (max 30% from one region) makes coordination harder
2. Selection is from $1,000+ stakers only (no cheap Sybils)
3. If even one of the 6 (3 primary + 3 backup) defects and submits fraud, Bob loses $200
4. Expected value: negative even with 90% corrupt verifiers, and that assumes all 6 are corrupt

The attack surface is massive. Bob needs to successfully bribe 6 unknown verifiers across multiple regions within an hour, and pay each more than $40, all to steal $100. It's not happening.

8.3 Client Griefs by Not Finalizing

Bob calls `claimTimeout()` after 1 hour (the challenge window). He gets payment + collateral + Alice's bond. Alice loses $55 to lock $200 for an hour. After 3 griefing attempts, Alice is banned. Griefing is -EV and limited.

8.4 Flash Loan Attack on Staking

Attacker borrows 50k CERTUS, stakes for insurance pool benefits, accepts job, returns loan. Mitigated with vesting requirements. Insurance pool participation requires staked CERTUS with lockup periods. Flash loans can't access these benefits.

8.5 Verifier Selection Manipulation

Would require controlling Arbitrum sequencer, frontrunning submissions, predicting blockhashes, and manipulating 11 entropy sources. Cost: $10k+ in MEV opportunity. Gain: -$200 (fraud still loses money even with controlled selection due to 2× collateral).

8.6 Gas Price Spikes Making Fraud Proofs Unprofitable

Fraud bounty is 20% of collateral. For a $50 job with 2× collateral, bounty is $40. Fraud proof costs $0.38 at current gas. Even at 100× gas ($38), it's still profitable. At realistic spikes (10× = $3.80), profit is $36.20. Still worth it.

8.7 Arbitrum Downtime

If Arbitrum is down, jobs can't be created or finalized. Executors don't lose deposits (deadlines extend). Users are affected like any Arbitrum dApp. Mitigation: Multi-chain deployment (Base, Optimism, zkSync).

8.8 Wasm Vulnerability

If a bug in Wasm implementations causes non-deterministic behavior, fraud proofs break. Mitigated by using battle-tested Wasm runtimes (Wasmtime, which powers Fastly edge compute) and restricting to deterministic subset. Edge cases would require governance to update validation rules.

9 The Roadmap (What We're Actually Building)

9.1 Phase 1: Mainnet Launch (Q1 2026)

- Deploy contracts on Arbitrum mainnet
- Launch with 20 initial verifier nodes (partners + team)
- Open verifier registration (public, $1,000 USDC minimum stake)
- Process initial production jobs (FFmpeg transcoding, SHA-256 hashing, image filters)
- Launch Directory service (centralized matchmaking for MVP)
- Target: 50+ active verifiers, 99% uptime, 1,000+ jobs/month

9.2 Phase 2: Token Launch & Scaling (Q2 2026)

- Launch CERTUS token on Arbitrum
- Enable veCERTUS staking (revenue share mechanism)
- Enable insurance pool staking (50% slashing coverage)
- Distribute 10M CERTUS airdrop to early adopters
- Integrate multiple entropy sources for verifier selection
- Publish SDK (TypeScript, Python, Rust)
- Target: 10,000 jobs/month, $500k monthly volume

9.3 Phase 3: Enterprise Adoption (Q3 2026)

- Launch enterprise features (priority execution, dedicated verifiers)
- Integrate with major cloud providers for hybrid deployments
- Expand Wasm module library (more ML models, codecs, algorithms)
- Launch verifier insurance pool
- Target: 100k jobs/month, $5M monthly volume

9.4 Phase 4: Decentralization (Q4 2026)

- Decentralize Directory service (P2P node discovery)
- Governance transition (DAO controls parameters)
- Multi-chain deployment (Base, Optimism for redundancy)
- Target: 500k jobs/month, $25M monthly volume, self-sustaining

10 What Success Looks Like

10.1 If This Works

Verifiable compute becomes a commodity. Developers stop trusting AWS or OpenAI to run their code correctly. They submit Wasm + inputs, get cryptographically verified outputs, pay $1 to $10 per job.

Artists provably generate NFTs according to claimed algorithms. Traders provably execute strategies as published. Lotteries are provably fair. ML models are provably run on correct inputs without tampering.

The trust layer moves from legal contracts and brand reputation to math and cryptoeconomics.

10.2 Market Size

Cloud compute is $750B per year. Most of that is databases, networking, storage, not compute. Pure compute (Lambda, Cloud Functions, Batch) is maybe $75B per year.

How much of that is deterministic, verifiable, and worth paying a premium for? Hard to say. Maybe 1% ($750M per year). Maybe 10% ($7.5B per year). At scale, we take 1 to 3% of job value as protocol fees. That's $7.5M to $225M per year in revenue at saturation.

10.3 The Real Opportunity

Verifiable compute doesn't just replace AWS. It's infrastructure for an entirely new computing paradigm that hasn't existed before. Here's what becomes possible:

10.3.1 Autonomous AI Agents You Can Actually Trust

Today, if someone launches an "AI trading bot" or "autonomous DAO agent," you're trusting them not to override it, rug pull, or frontrun the decisions. With Certus, the agent's code runs deterministically and verifiably. You can prove the bot executed exactly as programmed, with no human intervention. This unlocks AI agents managing billions in assets—something impossible today due to trust issues.

10.3.2 Decentralized Science That's Actually Reproducible

The replication crisis in science is real. Studies fail to reproduce because code has bugs, data preprocessing varies, or researchers cherry-pick results. With verifiable compute, every paper's computational work becomes reproducible by design. Submit the Wasm + data, get a cryptographic proof that output X came from input Y. No arguments, no "it works on my machine," no hidden parameters. This could fix a fundamental problem in science.

10.3.3 Fair Gaming Without Game Servers

Today, multiplayer games need centralized servers because you can't trust clients. But centralized servers can cheat (fudge RNG, manipulate outcomes for whales). With verifiable compute, game logic runs deterministically. Every player can verify that dice rolls, card shuffles, and loot drops followed the published algorithm. This enables actually fair gambling, esports tournaments with provable fairness, and play-to-earn games where the house can't cheat.

10.3.4 Smart Contracts That See the Real World

Right now, smart contracts are blind. They can't trigger on off-chain events (ML model predictions, video analysis, data processing) without trusting an oracle. With Certus, contracts can verify that off-chain computation happened correctly. A DeFi protocol can verify that a credit score model ran correctly. An insurance contract can verify that image analysis correctly identified a damaged car. This bridges blockchain and real-world compute without adding trust assumptions.

10.3.5 Verifiable Supply Chains

Imagine tracking pharmaceutical manufacturing where every step's computational checks are verifiable. Drug purity analysis, batch testing, quality control—all computed off-chain for cost, but verified on-chain for trust. No need to trust the manufacturer's lab equipment. The computation is provable.

10.4 The Actually Big Idea

Right now, "don't trust, verify" only applies to money moving between addresses. Everything else requires trust: trust in AWS, trust in OpenAI, trust in your cloud provider. We're extending "don't trust, verify" to general-purpose computation. That's the same conceptual leap Bitcoin made for money. Money used to require trusted banks. Now it doesn't. Compute currently requires trusted servers. Soon it won't.

These markets don't exist yet because there's no infrastructure. We're building the rails.

11 Why This Might Fail

Let's be honest about what could go wrong.

11.1 Adoption Risk

Developers might not care about verifiable compute. They trust AWS. They trust OpenAI. Decentralization is a nice-to-have, not a must-have. If nobody needs verification, nobody uses the protocol.

Counter: Crypto has $2T in value secured by trustlessness. DeFi proved there's demand for trust-minimized finance. The question is whether that extends to compute. We think it does, but it is not guaranteed.

11.2 Execution Risk

We're building three hard things: deterministic sandbox, fraud proof protocol, economic incentives. If any breaks, the whole system breaks. A bug in Wasm validation could allow non-deterministic code. A flaw in bisection could let executors cheat. A miscalibration in economics could make attack profitable.

Counter: We're not inventing new cryptography or consensus. We're combining proven pieces (Wasm, Arbitrum, collateral). The attack surface is smaller than a full blockchain. Still, "complex systems fail" is real.

11.3 Competition Risk

Arbitrum could build this into their stack. Chainlink could add compute verification to CCIP. A well-funded competitor with better execution could eat our lunch.

Counter: We're open source. If someone builds this better, that's fine—the world gets verifiable compute. We have first-mover advantage and deep understanding of the problem. That's worth something.

11.4 Regulatory Risk

Regulators might view verifiable compute as "unlicensed cloud computing" or "unregistered securities" (if CERTUS appreciates). Unclear jurisdiction (executor in US, client in EU, verification in Singapore).

Counter: We're not hosting data, just verifying computation. That's closer to Chainlink (which operates freely) than to AWS (which is heavily regulated). CERTUS is utility (like ETH), not equity. Still, regulation is unpredictable.

11.5 Black Swan Risk

Quantum computers break cryptographic assumptions. Wasm spec changes in backwards-incompatible way. Arbitrum rug pulls. Satoshi comes back and says "this is dumb."

Counter: We can't defend against unknown unknowns. We monitor, adapt, or fail.

12 Conclusion

Blockchains verify money. Certus makes them verify everything else.

We're not trying to replace AWS. We're not trying to decentralize all compute. We're building infrastructure for the specific case where verification matters more than cost or performance.

If you're running a web server, use AWS. It's faster and cheaper.

If you're generating a $1M NFT and want to prove it follows your algorithm, use Certus. The premium is worth it.

If you're running a trading bot with other people's money and want to prove you're not frontrunning, use Certus.

If you're building an on-chain AI agent and want to prove it executes as programmed, use Certus.

The tech is simple: deterministic Wasm, fraud proofs via bisection, collateral economics. The insight is combining these pieces in a way that makes verification cheap, fast, and trust-minimized.

But the implications aren't simple. If this works, if verifiable compute becomes as ubiquitous as verifiable money transfers, the trust assumptions underpinning most of modern computing start to break down. You won't need to trust that AWS ran your code correctly. You won't need to trust that an AI agent followed its programming. You won't need to trust that game servers are fair or that scientific computations are reproducible. You can verify all of it, cryptographically, for $1 to $10 per job.

That's not a small thing. It's the difference between "trust me, I ran the model correctly" and "here's a proof you can verify yourself that the model ran correctly." The former requires reputation, legal contracts, and brand equity. The latter requires math. As crypto has shown, math tends to win in the long run.

We're shipping mainnet in Q1 2026. If you want to run a verifier, build on the platform, or just poke holes in the design, reach out.

This paper will age poorly as we ship and reality hits. That's fine. Better to build and be wrong than to talk and be safe.

Evan W.
evan@certuscompute.com
@CertusCompute

All mistakes are mine.