BitLogic from Modulus: zkFOL Bitcoin

An Engineering Whitepaper for Formal Logic,
Zero-Knowledge, and Decentralized Finance on Bitcoin

by Mr. O' Modulus (2025), ModulusZK.io

November 20, 2025

Contents

1. Abstract

Bitcoin's original scripting language was designed with one overriding goal: security through simplicity. That simplicity, however, came at the cost of expressivity. Because Bitcoin Script cannot loop, maintain state, or execute arbitrary logic, it cannot host decentralized-finance primitives, automated contracts, or rich security policies directly on-chain.

This whitepaper introduces a mathematically grounded solution: **Modulus zkFOL Bitcoin**. It applies Dr. Murdoch Gabbay's finite-field arithmetisation of First-Order Logic (FOL) to Bitcoin, allowing all contract conditions to be expressed as formal logical predicates and compiled into polynomial commitments over a finite field. Each predicate is verified using succinct zero-knowledge proofs with bounded, deterministic cost.

The system initially operates as a Bitcoin Layer-2 network in which users lock BTC in provably reserved vaults and receive one-to-one wrapped tokens (`wBTC-FOL`) on the zkFOL chain. Within that layer, complex interactions; exchanges, lending, automated market-making; are verified purely by mathematics rather than executable code.

The long-term objective is to demonstrate that this model is efficient enough to justify a minimal Bitcoin soft-fork adding three opcodes; `OP_POLYEVAL`, `OP_FIELDVERIFY`, and `OP_COMMITCHECK`; enabling direct polynomial-proof verification on the base layer. In doing so, zkFOL aims to extend Bitcoin's capability while preserving its core design philosophy: simplicity, determinism, and predictable cost.

2. Background

2.1 The Limits of Bitcoin Script

Bitcoin Script is deliberately non-Turing-complete: a stack language with no loops, recursion, or unbounded data structures. This ensures every transaction terminates deterministically but forbids higher-level logic.

Script cannot:

- Store or reference persistent state between transactions,
- Execute iterative or recursive checks,
- Use external or dynamic data sources,
- Perform high-precision or floating-point arithmetic, or
- Implement multi-party logic without extensive manual branching.

Developers have long sought workarounds; federated sidechains, custodial bridges, or wrapped assets; but these weaken Bitcoin's native trust model. Meanwhile, smart-contract platforms such as Ethereum and Solana provide expressivity at the cost of determinism and increased attack surface (re-entrancy, gas volatility, consensus fragility).

The challenge is therefore conceptual: how to add expressive programmability to Bitcoin without compromising its deterministic execution model.

2.2 First-Order Logic as a Contract Language

First-Order Logic (FOL) offers a precise mathematical language for expressing relationships as quantified predicates:

$$\forall x\, \exists y.\ \phi(x, y),$$

where ϕ is a logical formula composed of atomic relations and Boolean connectives.

Dr. Gabbay's 2024 framework shows that such predicates can be *arithmetised* into multivariate polynomials over finite fields $\mathbb{F}_p$. For every ϕ there exists a polynomial P_ϕ such that

$$\phi \text{ is true } \iff P_\phi = 0.$$

Logical connectives map cleanly to algebraic operations:

$$\phi \wedge \psi \longrightarrow P_\phi \cdot P_\psi,$$
$$\phi \vee \psi \longrightarrow P_\phi + P_\psi,$$

Quantifiers are evaluated over a bounded subset $H \subset \mathbb{F}_p$, making verification finite and deterministic:

$$\forall x \in H :\ P_\phi(x) = 0, \qquad \exists x \in H :\ \sum_{x \in H} P_\phi(x) = 0.$$

Checking a predicate thus reduces to verifying that a single polynomial evaluates to zero at a randomly chosen $a \in \mathbb{F}_p$. By the Schwartz–Zippel lemma, the false-acceptance probability is $\leq \deg(P)/p$, negligible when $\deg(P) \ll p$ and $p > 2^{256}$.

2.3 From Logic to Zero-Knowledge Proofs

To provide compact, privacy-preserving verification, zkFOL employs the Zinc/Zip proving stack. The compilation pipeline is:

$$\text{FOL Spec} \to \text{Polynomial IR} \to \text{Lifted Integer Constraints } (\text{R1CS}_L)$$
$$\to\ \text{PIOP Projection} \to \text{Zip Polynomial Commitment Scheme} \to \text{Succinct SNARK Proof.}$$

The prover commits to the coefficients of P_ϕ, performs random evaluations, and produces a zero-knowledge proof that $P_\phi(a) = 0$ without revealing a or the internal structure of ϕ. Verification requires only a single field evaluation and group operation; well within Bitcoin's potential opcode budget.

This synthesis of formal logic, finite-field algebra, and zero-knowledge commitments transforms Bitcoin Script from a rigid stack language into a provable logical system with predictable computational cost.

3. System Architecture

3.1 High-Level Overview

The zkFOL Layer is a two-tier Bitcoin extension:

1. **Layer 1 (Bitcoin):** Holds actual BTC in multisignature vaults governed by transparent proof-of-reserve logic.
2. **Layer 2 (zkFOL Chain):** Hosts logical contracts expressed in FOL and proven via finite-field SNARKs.

When BTC is locked on Layer 1, an equal amount of `wBTC-FOL` is minted on Layer 2. All Layer-2 transactions; swaps, loans, vault interactions; are accompanied by zkFOL proofs that can be publicly verified without revealing private data. Withdrawals release the corresponding BTC once the latest proof state is validated.

3.2 Execution Flow

1. **Authoring:** Developers define contract logic as FOL predicates. For example:

   Notation. We use the rule form $\mathit{head} \leftarrow \mathit{body}$ to mean "the head predicate holds if the body's conditions are satisfied," following Horn-clause and logic-programming convention.

   $$\mathit{valid\_swap}(s, s') \;\leftarrow\; \exists k.\ (s'.A = s.A - k) \wedge (s'.B = s.B + f(k, s)) \wedge (s.A \cdot s.B = s'.A \cdot s'.B).$$

2. **Compilation:** The predicate is translated into a polynomial $P(x_1, \ldots, x_n)$ and constraints.
3. **Proof Generation:** Off-chain validators compute a zero-knowledge proof showing the predicate holds.
4. **Verification:** The proof and commitment are posted to the zkFOL chain, where $P(a) = 0$ is checked.

3.3 Key Components

- **Bridge Vaults:** Bitcoin multisig addresses holding reserves and emitting Merkle-auditable proofs.
- **zkFOL Validator Set:** Nodes generating and verifying polynomial proofs for fees.
- **Wrapped Asset Token (wBTC-FOL):** A 1:1-backed token representing claimable BTC.
- **Logical Contract Compiler:** Converts FOL specifications into arithmetic circuits.
- **Proof Verifier Module:** Lightweight verifier compatible with both zkFOL and potential Bitcoin opcodes.

3.4 Security and Determinism

Unlike Turing-complete systems, zkFOL contracts are stateless and deterministic: each transaction is an independent proof of a logical condition. There is no global mutable state, eliminating reentrancy, recursion, and non-termination risk while retaining expressivity through formal logic.

3.5 Consensus and Data Availability

The zkFOL chain can function as an optimistic rollup or sidechain anchored to Bitcoin. Proof commitments are periodically published to Bitcoin for settlement finality. Any participant can recompute polynomial proofs from on-chain data, ensuring transparency and censorship resistance.

4. Bitcoin-Native DeFi

The zkFOL Layer is designed to make Bitcoin the secure base asset for an entire decentralized-finance (DeFi) stack. By expressing all logic as formal predicates and verifying them as finite-field proofs, the system achieves Ethereum-level expressivity without leaving Bitcoin's deterministic framework.

Each class of application is defined by:

1. A logical specification (FOL predicate),
2. Its corresponding polynomial representation, and
3. An operational flow (proof generation and verification).

4.1 Decentralized Exchange (DEX)

The simplest example of programmable liquidity is the constant-product invariant between two assets. In conventional DeFi systems this is expressed as $x \times y = k$. Within zkFOL, this becomes a logical predicate over initial and final states s and s':

$$\mathit{valid\_swap}(s, s') \leftarrow \exists k.\ (s'.A = s.A - k) \wedge (s'.B = s.B + f(k, s)) \wedge (s.A \cdot s.B = s'.A \cdot s'.B).$$

Polynomial Translation. Each conjunct becomes an equality constraint. The conjunction is represented multiplicatively:

$$P_{\text{swap}}(A, B, A', B', k) = (A' - A + k)\,(B' - B - f(k, s))\,(A \cdot B - A' \cdot B').$$

Verification requires that $P_{\text{swap}} = 0$ at a random evaluation point.

Operational Flow.

1. A trader submits an order $(\Delta A, \Delta B)$.

2. A validator computes k and generates a zkFOL proof that the invariant holds.
3. The verifier checks the proof; reserves update deterministically.
4. Fees are introduced via a term $\text{fee} = \alpha|\Delta A|$ encoded inside $f(k, s)$.

Because proofs are succinct and privacy-preserving, order size and counterparties remain hidden.

4.2 Lending and Collateralized Loans

A lending protocol can be expressed algebraically without global state.

Predicate.

$$\mathit{loan}(tx) \leftarrow \mathit{sig}(\mathit{borrower}) \wedge \mathit{collateral}(tx) \geq \rho \cdot \mathit{debt}(tx) \wedge \mathit{time} \leq \mathit{expiry},$$

where ρ is the required collateralization ratio.

Polynomial.

$$P_{\text{loan}} = (\mathit{sig})(\mathit{collateral} - \rho \cdot \mathit{debt})(\mathit{time} - \mathit{expiry}).$$

When $P_{\text{loan}} = 0$ under evaluation, the borrowing condition holds.

Repayment is another predicate:

$$\mathit{repay}(tx) \leftarrow \mathit{sig}(\mathit{borrower}) \wedge \mathit{payment} \geq \mathit{debt}(1 + rt),$$

producing

$$P_{\text{repay}} = \mathit{payment} - \mathit{debt}(1 + rt).$$

Operational Flow.

1. The borrower locks collateral BTC in a vault.
2. Validators mint debt tokens once $P_{\text{loan}} = 0$.
3. Interest accrues deterministically as a function of rt.
4. Repayment proofs burn the debt token and release collateral.

Each transaction independently satisfies its local invariants; no contract state is retained on-chain.

4.3 Automated Market Maker (AMM)

Liquidity pools generalize the DEX invariant to multiple assets and curves:

$$\mathit{amm}(s, s') \leftarrow \exists \Delta_A, \Delta_B.\ (A' = A + \Delta_A) \wedge (B' = B - \Delta_B) \wedge (A + \Delta_A)(B - \Delta_B) = k.$$

Polynomial.

$$P_{\text{amm}} = (A' - A - \Delta_A)(B' - B + \Delta_B)((A + \Delta_A)(B - \Delta_B) - k).$$

Liquidity Provision. Depositors prove proportional share increases:

$$\mathit{provide}(tx) \leftarrow \mathit{sig}(\mathit{provider}) \wedge \frac{\Delta_A}{A} = \frac{\Delta_B}{B}.$$

This compiles into the equality polynomial:

$$P_{\text{provide}} = (\Delta_A B - \Delta_B A).$$

Yield. Protocol fees α are distributed by:

$$\mathit{reward}_i = \alpha \cdot \frac{\mathit{share}_i}{\sum \mathit{share}}.$$

zkFOL verifies that the total distributed rewards equal collected fees, conserving value without revealing balances.

5. Enhanced Security Features

Security logic is where zkFOL demonstrates its strongest advantage. Complex spending paths that would require multiple conditional branches in Script become concise algebraic predicates.

5.1 Vaults

A vault may enforce multiple authorized spend paths:

$$\begin{aligned}\mathit{vault}(tx) \leftarrow (\mathit{delay}(tx) > 48h \wedge \mathit{signed}(tx, \mathit{owner})) \vee\ & (\mathit{emergency}(tx) \wedge \mathit{multisig}(tx, \mathit{trustees})) \\ \vee\ & (\mathit{timeout}(tx, 1y) \wedge \mathit{recovery}(tx)).\end{aligned}$$

Each disjunct becomes an additive term within one polynomial:

$$\begin{aligned}P_{\text{vault}} = (\mathit{delay} - 48h)(1 - \mathit{signed}_{\mathit{owner}}) \\ + (1 - \mathit{emergency}\, \mathit{multisig}_{\mathit{trustees}}) \\ + (\mathit{time} - 1y)(1 - \mathit{recovery}).\end{aligned}$$

Note. The additive form encodes a logical disjunction (at least one branch). If mutual exclusivity is required, add disjo

Verification of $P_{\text{vault}} = 0$ ensures that at least one valid branch is satisfied; disjointness constraints can enforce exclusivity.

Use-Cases.

- Time-locked savings with emergency recovery,
- Institutional custody where trustees can override in emergencies,
- Decentralized inheritance with fallback keys.

Additional clauses are appended by multiplying new terms; no need to rewrite base logic.

5.2 Dynamic Spending Limits

Account-based systems enforce spending limits through mutable state; zkFOL replaces this with verifiable proofs.

Predicate.

$$\mathit{limit}(tx) \leftarrow (\mathit{amount} \leq \mathit{daily\_limit}(\mathit{owner})) \vee \mathit{multisig}(tx).$$

Evolution Rule.

$$\mathit{limit}_{t+1} = \mathit{limit}_t + g(\mathit{time}),$$

where $g(t)$ is a deterministic increment function encoded in the proof.

Implementation. Owners hold a cryptographic commitment $C_t = H(\mathit{limit}_t)$. To spend, they prove the amount satisfies the committed limit and update to C_{t+1}. No external storage is required; commitments are carried in witness data.

5.3 Fraud Proofs and Watchtower Protocols

Fraud detection can be expressed as a logical predicate:

$$\mathit{fraud}(tx) \leftarrow \neg \mathit{authorized}(tx) \wedge \mathit{amount}(tx) > \mathit{threshold}.$$

Watchtowers monitor blocks and attempt to produce $\mathit{fraud}(tx)$ proofs. A valid proof rewards the reporter with a portion of confiscated funds, turning monitoring into a self-incentivized market.

Predicate Structure.

1. Authorization check: detect missing or invalid signature commitments.
2. Threshold check: verify $\mathit{amount} - \mathit{threshold} > 0$.
3. Penalty rule: confirm slashed funds transfer to reporter.

All components are verified through a single algebraic constraint system, enabling non-interactive dispute resolution.

6. Complex Inheritance Schemes

Inheritance contracts combine temporal and authorization logic:

$$inheritance(state) \leftarrow notarized(heir) \wedge (time > death\_timestamp) \wedge \neg revoke(owner).$$

Polynomial Translation.

$$P_{\text{inherit}} = notarized \cdot (time - death) \cdot (1 - revoke).$$

If $P_{\text{inherit}} = 0$, control transfers automatically to the heir; no executors or custodians required.

Multi-Generation Design. Heir contracts can themselves contain nested predicates referencing prior proofs:

$$heir2(tx) \leftarrow verified(P^{(1)}_{\text{inherit}}) \wedge notarized(heir2).$$

Each generation composes algebraically over the previous one, enabling perpetual digital estates with mathematically enforced succession.

7. Verification, Security, and Privacy

The zkFOL verification model ensures every transaction and contract is mathematically sound, deterministic in cost, and publicly verifiable without external trust.

7.1 Finite-Field Verification Model

All contract logic compiles into a polynomial

$$P(x_1, \ldots, x_n) \in \mathbb{F}_p[x_1, \ldots, x_n].$$

Verification requires evaluating this polynomial at a random challenge point $a \in \mathbb{F}_p$ and checking that $P(a) = 0$.

By the Schwartz–Zippel lemma, if P is not identically zero, the chance of false acceptance is bounded by $\deg(P)/p$. For $p > 2^{256}$ and $\deg(P) \ll p$, this probability is negligible.

Each FOL connective translates as:

$$\begin{aligned}
\phi \wedge \psi &\rightarrow P_\phi \cdot P_\psi, \\
\phi \vee \psi &\rightarrow P_\phi + P_\psi, \\
\neg\phi &\rightarrow 1 - P_\phi, \\
\forall x.\phi(x) &\rightarrow \prod_{x \in H} P_\phi(x), \\
\exists x.\phi(x) &\rightarrow \sum_{x \in H} P_\phi(x),
\end{aligned}$$

where $H \subset \mathbb{F}_p$ is a bounded evaluation domain. This mapping provides a complete algebraic semantics for FOL over finite fields, aligning with Gabbay's formal construction.

Deterministic Evaluation. All arithmetic operations occur in modular arithmetic within $\mathbb{F}_p$, ensuring bounded runtime and consistent results across all verifiers; critical for Bitcoin consensus compatibility.

7.2 Soundness and Determinism

Each zkFOL proof depends solely on data inside the transaction and a deterministic challenge derived from the Bitcoin block header:

$$a = H(block\_header||txid||nonce).$$

This random-oracle-style derivation guarantees unpredictability and prevents replay or grinding attacks. A proof valid in one block becomes invalid in another, embedding time and context into each verification.

Because every step is deterministic, all nodes reach identical conclusions from identical data; making zkFOL compatible with consensus-layer verification.

7.3 Zero-Knowledge Privacy

zkFOL integrates zero-knowledge proving via the Zip polynomial commitment scheme to preserve user privacy. The verifier confirms correctness without learning private values or intermediate computations.

Formally, for a commitment scheme

$$C : \mathbb{F}_p^n \to G,$$

the verifier checks

$$\text{Verify}(C(P), a, y) \Rightarrow (P(a) = y = 0).$$

Only the evaluation point a and the commitments are revealed; polynomial coefficients remain secret. Thus traders, lenders, and vault owners can prove invariants without disclosing balances, durations, or transaction details.

7.4 Resistance to Attacks

- **Replay Attacks:** Prevented by including block headers in the challenge hash.
- **Front-Running:** Mitigated because order contents remain hidden until proof verification.
- **Brute-Force Search:** Infeasible due to the magnitude of p.
- **Invalid Proof Forgery:** Reduced to the hardness of the underlying SNARK commitment scheme.

The overall security assumption is no weaker than Bitcoin's elliptic-curve cryptography itself.

8. Economic Model

To remain sustainable, zkFOL must generate predictable revenue and align incentives among participants. Unlike inflationary models, zkFOL economics are fee-based and proportional to network usage.

8.1 Revenue Sources

1. Verification Fees. Each proof submission pays a fee proportional to polynomial size:

$$fee = \alpha \times \text{size}(P),$$

where α is the fixed cost per polynomial term. These fees fund validators and maintainers.

2. Bridge Fees. A small percentage (e.g. 0.1%) is charged on peg-in and peg-out BTC transactions. Collected fees flow into a DAO treasury that supports validators and research.

3. Validator Rewards. Validators earn transaction fees for generating and verifying proofs. Market competition ensures efficiency and keeps fees low.

4. Governance and Treasury. A zkFOL DAO can allocate treasury funds toward insurance pools, grants, or protocol safety mechanisms. All treasury movements are governed by on-chain zkFOL predicates verifying vote outcomes mathematically.

8.2 Network Sustainability

zkFOL avoids gas volatility by bounding every transaction's computation. Because all logic compiles into fixed-degree polynomials, cost remains constant over time.

Multiple proofs can be aggregated for scalability:

$$P_{\text{agg}} = \sum_{i=1}^{n} w_i P_i,$$

Aggregation applies only when constituent circuits share identical constraint structure and commitment keys.

where aggregation is valid for circuits sharing identical structure. One verification can thus confirm hundreds of batched transactions, reducing per-user cost and matching rollup-level throughput.

Proof-of-Reserve. The bridge's solvency is maintained through Merkle-auditable BTC reserves. All `wBTC-FOL` supply is cryptographically tied to on-chain Bitcoin holdings.

8.3 Security Budget and Long-Term Viability

Bitcoin's future security depends on transaction fees as block rewards diminish. zkFOL enhances this by introducing new high-value transaction types whose settlement commitments are periodically published to Bitcoin. Each zkFOL batch therefore generates additional miner fees and strengthens Bitcoin's long-term economic base.

In this sense, zkFOL is not merely a layer on top of Bitcoin; it acts as an economic catalyst reinforcing Bitcoin's sustainability.

9. Roadmap, Alignment, and Conclusion

9.1 Deployment Phases

Phase I ; Prototype and Proof of Concept (2025–2026): Implement the zkFOL compiler and verifier stack. Demonstrate small predicates such as multisig, vaults, and swaps. Benchmark polynomial evaluation costs in Bitcoin Script simulations.

Phase II ; Layer-2 Network and Developer Toolkit (2026): Launch a testnet with wrapped BTC, validator participation, and initial DeFi primitives (DEX, lending, AMM, yield). Release a developer SDK that compiles logical contracts to proofs automatically.

Phase III ; Ecosystem Growth and DAO Governance (2026–2027): Form the zkFOL DAO, integrate with wallets and custodians, and deploy enterprise-grade vaults, insurance, and on-chain governance modules.

Phase IV ; Soft-Fork Integration (2027 and beyond): Propose minimal Bitcoin opcodes: `OP_POLYEVAL`, `OP_POLYCOMMIT`, and `OP_FIELDVERIFY`. Once adopted, all zkFOL predicates can verify directly on Bitcoin's base layer, eliminating the need for wrapped assets.

9.2 Alignment with Bitcoin Philosophy

Bitcoin's resilience stems from simplicity and predictability. zkFOL adheres to those principles through five tenets:

1. **Simplicity:** Core consensus remains untouched; complexity moves off-chain.
2. **Security:** Contracts cannot loop or recurse; every condition terminates in polynomial time.
3. **Opt-In Complexity:** Users engage with zkFOL voluntarily; non-participants experience no protocol change.
4. **Backward Compatibility:** Existing Bitcoin scripts remain valid; zkFOL adds optional capability.
5. **Predictable Cost:** Verification scales linearly with formula size; never exponentially.

Through these design principles, zkFOL expands Bitcoin's expressivity while preserving its conservative ethos.

9.3 Broader Economic and Philosophical Impact

Reintroducing expressive logic atop Bitcoin invites a new wave of developer innovation. For the first time, decentralized exchanges, credit markets, and autonomous vaults can be constructed entirely within Bitcoin's security model. Capital efficiency increases, trust centralizes back to Bitcoin, and the network evolves into an integrated foundation for global finance.

Beyond finance, the same mathematical framework can encode identity proofs, supply-chain attestations, and governance mechanisms; each verifiable by succinct algebra rather than opaque code execution. In this way, zkFOL transforms Bitcoin from a passive store of value into an active substrate for verifiable computation.

9.4 Conclusion

Modulus zkFOL Bitcoin unites three mathematical disciplines; First-Order Logic, Finite-Field Algebra, and Zero-Knowledge Proofs; into a coherent model for secure and private programmability. It proves that rich smart-contract functionality need not require Turing completeness, only a reinterpretation of logic as algebra.

By shifting computation off-chain and verifying only succinct proofs on-chain, zkFOL preserves Bitcoin's determinism while enabling expressive financial and governance systems. It offers a path toward a future Bitcoin where complexity is optional, security is provable, and programmability emerges not from code execution but from mathematics itself.