USENIX Security '26 Artifact Appendix: Bridging Bitcoin to Second Layers via BitVM2

A Artifact Appendix

This artifact appendix provides a complete evaluation roadmap for the BitVM2 implementation. It includes BITVM2-CORE (arbitrary computation engine) and BITVM2-BRIDGE (first light-client-based Bitcoin bridge). The artifact demonstrates mainnet validation via transactions executed June 2025, with reproducible performance characteristics on regtest/signet networks.

A.1 Abstract

BitVM2 enables arbitrary computation on Bitcoin through SNARK verification using optimistic computation. This artifact includes: (1) a Groth16 SNARK verifier compiled to Bitcoin Script (∼900MB, chunked to <4MB pieces); (2) complete peg-in/peg-out bridge protocol; (3) 1-of-n trust minimization through covenant emulation; (4) mainnet validation via June 2025 transactions; and (5) performance characteristics with <8 hour dispute resolution and <0.15 BTC fees. All major claims are reproducible on Bitcoin regtest/signet.

A.2 Description & Requirements

This section describes the environment required to evaluate the BitVM2 artifact.

A.2.1 Security, privacy, and ethical concerns

Bitcoin regtest mode: The artifact runs Bitcoin Core in regtest (regression test) mode, creating a private local blockchain. No real funds are used during evaluation.

Resource usage: Building the Groth16 verifier requires substantial memory (16GB+ RAM). Some tests generate large files (up to 4MB Bitcoin Script files).

Network activity: Requires internet connectivity to download dependencies (Rust crates, Risc0 toolchain). Bitcoin Core will bind to network ports for the regtest node.

A.2.2 How to access

Repository: https://github.com/bitvm/bitvm

Permanent archive: https://doi.org/10.5281/zenodo.17949747

The artifact is permanently archived on Zenodo with DOI 10.5281/zenodo.17949747, ensuring long-term availability as required for the Artifacts Available badge.

A.2.3 Hardware dependencies

Minimum requirements:

- **CPU**: x86_64 or ARM64 (Apple Silicon supported)
- **RAM**: 16GB minimum, 24GB recommended
- **Disk**: 10GB free space

Tested configurations: Ubuntu 22.04 (x86_64), macOS (ARM64)

A.2.4 Software dependencies

Required: Rust 1.70+ (per `rust-toolchain.toml`), Cargo, Git, Bitcoin Core 25.0+, Risc0 toolchain.

Optional: Docker (containerized regtest), CUDA toolkit (NVIDIA GPUs).

All dependencies are available via standard package managers. Detailed installation in Section 1.3.1.

A.2.5 Benchmarks

Bitcoin blockchain data generated in regtest mode. No external datasets required. Mainnet validation uses publicly accessible Bitcoin mainnet transactions from June 2025.

A.3 Set-up

A.3.1 Installation

Step 1: Clone repository

```
git clone https://github.com/bitvm/bitvm
cd bitvm
```

Step 2: Build (requires 16GB+ RAM)

```
cargo build --release
```

Step 3: Install Bitcoin Core

```
cd regtest && ./install.sh && cd ..
```

Step 4: Configure (edit with RPC credentials)

```
cp .env.sample .env
```

Step 5: Install Risc0 toolchain

```
cargo install cargo-risczero
cargo risczero install
```

A.3.2 Basic Test

Test 1: Verify build

```
cargo test --lib
```

Expected: Unit tests pass (some may be ignored by default for performance reasons).

Test 2: Start Bitcoin regtest

```
cd regtest && ./start.sh
bitcoin-cli -regtest getblockchaininfo
```

Expected: Shows `"chain": "regtest"` and `"blocks": 0`.

Test 3: Run basic script tests

```
cargo test --release u32
```

Expected: Tests for u32 Bitcoin Script primitives pass.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): BITVM2-CORE enables arbitrary computation on Bitcoin by compiling a Groth16 SNARK verifier to Bitcoin Script (∼900MB), chunked into 379 pieces (90% <3MB). Proven by (E1); described in Abstract & Section 3; results in Section 7.

(C2): Complete peg-in/peg-out bridge protocol with dispute resolution. Proven by (E2), (E3), (E4); described in Section 5; transaction graphs in Figure 1.

(C3): Trust minimized to 1-of-n (existential honesty) via covenant emulation. Proven by (E5); described in Section 2 & Figure 1.

(C4): Production-level implementation validated on Bitcoin mainnet (June 3, 2025). Proven by (E6); described in Abstract & Section 7.

(C5): Performance: <8h dispute, <0.15 BTC fees, 90% scripts <3MB. Proven by (E1), (E6); results in Section 7, Figures 8-9.

A.4.2 Experiments

(E1): BITVM2-CORE Functionality [5 min + 1GB disk]
Verify Groth16 SNARK verifier compiles to Bitcoin Script and is chunked appropriately.
Preparation: `cd bitvm`
Execution: `cargo test --release groth16 && cargo test --release chunk`
Results: Tests pass. Groth16 ∼900MB Script. 379 chunks, 90% <3MB, range 500KB-3.99MB.
Supports: (C1), (C5)

(E2): Peg-in Protocol [5 min]
Verify successful peg-in (deposit) to bridge.
Preparation: Ensure regtest running: `cd regtest && ./start.sh`
Execution: Follow `DEMO_INSTRUCTIONS.md` Section: "Peg-in Demo"
1. Generate deposit address
2. Fund deposit transaction
3. Confirm deposit on regtest
4. Verify bridge accepts deposit

Results: Deposit confirms in regtest. Bridge records successfully. Verify via Esplora (localhost:5000).
Supports: (C2)

(E3): Peg-out Happy Path [5 min]
Verify successful peg-out (withdrawal) with honest operator.
Preparation: Complete peg-in from (E2).
Execution: Follow `DEMO_INSTRUCTIONS.md` Section: "Peg-out Happy Path"
1. User initiates peg-out request
2. Operator processes withdrawal
3. Confirm peg-out transactions on regtest

Results: All transaction types confirm: Kick-off, Take, Peg-out. User receives funds. Tx sizes <17 vkB.
Supports: (C2), (C5)

(E4): Dispute Resolution [10 min]
Verify dispute mechanism penalizes dishonest operator.
Preparation: Complete peg-in from (E2).
Execution: Follow `DEMO_INSTRUCTIONS.md` Section: "Peg-out Unhappy Path"
1. Operator submits incorrect claim
2. Verifier challenges the claim
3. Operator posts Assert transaction
4. Verifier posts Disprove transaction

Results: Challenge mechanism works. Dispute resolves within timeout. Dishonest operator penalized. Assert & Disprove txs confirm.
Supports: (C2), (C5)

(E5): Trust Model Verification [10 min]
Verify 1-of-n trust minimization through code review.
Preparation: None (code review only).
Execution:
1. Review `bridge/src/contexts/` directory
2. Examine presigned tx structure in `bridge/src/transactions/`
3. Verify signer role (active only during setup)
4. Verify operator role (single honest party ensures liveness)
5. Verify challenger role (permissionless)

Results: Covenant emulation via presigned txs confirmed. Signers only active during setup. Single honest signer prevents theft. Single honest operator ensures liveness. Anyone can challenge.
Supports: (C3)

(E6): Mainnet Validation [5 min]

Verify actual Bitcoin mainnet transactions from June 2025.

Preparation: None (verification only).

Execution: Verify transactions via `mempool.space`:

1. Claim: `6b096f35...d4d147a`
2. Challenge: `49f3425b...5d80581`
3. Assert: `32129c9b...08711f`
4. Disprove: `8ecfefd4...510249`

Results: All txs confirmed on mainnet. Settlement: 42 blocks (7h 36min). Total cost: 14.9M sat (∼$16k). Largest tx: ∼4MB. Matches claims: <8h, <0.15 BTC.

Supports: (C4), (C5)

A.5 Notes on Reusability

Extending: Custom circuits (modify `bitvm/src/groth16/`), different networks (edit `.env`), reuse Script primitives (`bitvm/src/`), adjust bridge parameters (`bridge/src/`).

Limitations: On-chain light client (Section 4) NOT implemented. Large txs (>400kB) require custom relay infrastructure.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at `https://secartifacts.github.io/usenixsec2026/`.