Digging into BitVM 2

Antoine Riard, Bitcoinology, January 2025

About: ariard

- bitcoin protocol hacker ~2018 (base-layer and lightning)
- managing partner @ thelab31.xyz (R&D / security consulting “boutique”)
- areas of research interest: protocol security and bitcoin scalability
  - cross-layer mempool issues (e.g mempoolfullrbf)
  - lightning: time-dilation, dust-inflation and pinning attacks
  - coinpool and payment pools research
- privacy note: no photo thanks

About: Socratic talk

- the talk aims to be a dialogue asking and answering questions with the audience
- i’m assuming some Bitcoin 101 with blockchain fundamentals (Script, Tx, etc)
- if not familiar about technical basics do not hesitate to ask what does it mean
- if you have advanced remarks on the subjects, do not hesitate to grab the mic

The talk sections

I) Reminder on BitVM

II) BitVM 2: Building Blocks

III) BitVM 2: A Protocol Walkthrough

IV) BitVM 2: Use-Cases on top of the Protocol

V) BitVM 2: Protocol Limitations

A reminder on BitVM

- a “new” computing paradigm to express Turing-complete Bitcoin contracts
- fundamental innovation”: logic gate commitment in today’s Bitcoin Script
- promises: chess games, validity proofs verifications, alt-chain bridges
- *disclaimer*: my understanding of the paper material only
  - it might be imperfect, paper not very complete

The BitVM trick: Bit Value Commitment

- emulating the OP_BITCOMMITMENT with sequence of OP_HASH160
- each hash commit to a bit value, either *0* or *1*.
- emulating OP_NAND is possible too with OP_BOOLAND + OP_NOT
- by combining OP_NAND + OP_BITCMT, *optimiscally verifiable* logical gate

What is optimistic verification as a model ?

- Security model originally coming from the alt-chain and rollups-land.
- Computation of interest among second-layer counterparties moved off-chain.
- Digest submitted on-chain by a *prover* when the computation is over.
- *Challenge period* during which the *verifier* can contest the *digest validity*

What the heck is BitVM2 ?

- a significant improvement on the original BitVM design
- fundamental idea: still the same, express *any* computation on the blockchain
- main innovation proposed: *compressing* the computation with a SNARK verifier
- objective: *practically fits* the circuit of the computation within chain limits

Building Blocks: Lamport Signature

- a significant improvement on the original BitVM design
- a hash-based cryptographic scheme known since the end of the 70’s
- F: K -> V where F is a one way whose domain is the set of keys
- *public key* for a 1-bit data item : F(k.i), *signature*: k.i, where i *message* bit index

Building Blocks: Signature-based Covenants

- *covenant*: the scriptpubkey of a UTXO restricts the spending transaction
    - yes, we have already covenants in BTC, e.g CHECKLOCKTIME_VERIFY
- script-based covenants and sigs-based covenants: *immutability* as a distinction
    - *conjecture*: one cannot prove all copies of a private key have been deleted
- technique known in BTC since micropayment channels in ~2012
- multi-sig to constraint a spending tx under a form negotiated by counterparties

Building Blocks: Taproot Tree

- merkelized alternative tree of scripts, a Bitcoin Script extension since 2021
- basically, for 4 leaves a Merkle tree of: h(s_1) + h(s_2) + h(s_3) + h(s_4)
- PT2R *script-path spend*, block limit only on script size (i.e MAX_SCRIPT_SIZE)
- script-path spend have an *absolute limit* of 128 elements (BIP341)

Building Blocks: SNARG verification

- *SNARGs*: succinct non-interactive arguments in the pre-processing model
- A proof system to verify untrusted statements expressed as boolean circuits
- A triple of algorithms (G, P, V), with *G key generator*, *P prover*, *V verifier*
- Properties: *completeness*, *soundness*, *efficiency*

Building Blocks: Challenge-Response & Timelock

- Bitcoin Script have timelocks: nLocktime and nSequence-based (BIP68)
- in the context of Bitcoin L2s, can be used for challenge-response flow
- e.g using a nLocktime until block height 100, tx cannot be included
- challenge-response flow: to *correct* Alice's state, Bob has until block 100

BitVM 2 : 3 On-chain Protocol Phases

- *Setup* phase: participants agree on an off-chain computation C
- *Execution* phase: operator (i.e the prover) executes the off-chain computation
- *Commit and Challenge* phase:
  - 1) operator submit on-chain an *assertion* proof
  - 2) challenger counter-submit on-chain a *disprove* proof
- *Terminal* phase: operator gets the set-up *payout* or challenger wins the reward

BitVM 2 : The Setup Phase I

- set of participants split the computation C in N sub-steps
- each sub-step is a serie of instructions or logical gates of the computation
- for each computation sub-step, a Lamport pubkey is generated
- each pair of gates and pubkey committed in the locking script’s Taproot tree

BitVM 2 : The Execution Phase

- one participant is performing the computation C off-chain
- e.g resolving a sudoku that has been translated as the computation C
- the participant keeps the list of resulting states for each sub-step
- this list of resulting states is *the trace* to be proven on-chain

BitVM 2 : The Commit Phase

- the participant publishes on-chain *the trace* by spending the deposit tx
- the commitment is done by revealing the Lamport private keys, i.e the sigs
- this publication is done through the assertion tx witness
- *any-one* (in theory) can challenge the prover as the commitment is public

BitVM 2 : The Challenge Phase I

- any participant with knowledge of the *locking Taproot tree* can challenge`
- to be successful, a challenger should prove that a logical sub-step is incorrect
- the challenger picks a sub-step at index R formalizes as S.j <- F(S.i) where S state
- all the sub-steps states have been provided by the prover in the previous phase

BitVM 2 : The Challenge Phase II

- any participant with knowledge of the *locking Taproot tree* can challenge`
- to be successful, a challenger should prove that a logical sub-step is incorrect
- the challenger picks a sub-step at index R formalizes as S.j <- F(S.i) where S state
- all the sub-steps states have been committed in the previous phase

BitVM 2 : The Terminal Phase

- if the challenger proves to the Bitcoin Script, 2 statements
- *1st statement*: the S.j <- F(S.i) with given S.i is incorrect
- *2nd statement*: “I know a revealed commitment for S.i”
- If 2 statements good, the challenger wins the reward, or after T the operator

BitVM 2: The Use Cases

- participants can emulate practical computation with BitVM 2 protocol
- e.g an alt-chain bridge, where funds peg-out are the proven computation
- e.g a proof of execution correctness of a cloud virtual machine

BitVM challenges #1: circuit scale ?

- taproot tree size limit you can encode in PT2R (see bip341)
- witness growth scale with the circuit complexity
- 32 bytes * 128 = 4096 bytes at 1 sat / virtual bytes - this is *likely* practical
- there is still an unknown on the off-chain computation max circuit complexity

BitVM challenges #2: fee fault-tolerance

- chainspace beefy witness for the *Commit* and *Challenge* phases
- no guarantee of stable network mempool feerates during whole C&C phase
- any challenger might have to provision sufficient fees
- asymmetry among operators and challengers on the verification *timing*

BitVM challenges #3: “challenge” DoS

- ideally efficient sampling techniques to verify circuit execution in minimal steps
- counterparty cannot engage anymore in *correct-yet-lengthy* verification steps
- circuit size and max depth of execution “fused” at setup

BitVM challenges #4: pre-signed sequence txn

- all the permutations of the circuit gates are not pre-committed anymore

- the state is carried through the *Lamport pubkey / signatures*

- data / code separation covenant *appears* to solve this

- is the usage of Lamport sig scheme that way *sound* and *publicly verifiable* ?

Thanks to Bitcoinology!