MIT Digital Currency Initiative and the University of Brasilia presents

Cryptocurrency Design and Engineering

Lecture 13': BitVM and Bitcoin rollups
Taught by: Trey Del Bonis
Date: 10/21/2025
MAS.S62

Bitcoin rollups

Trey D – Alpen Labs

$ whoami

- Thinking about Lightning and blockchain scaling since 2018
- Preliminary work on what a Bitcoin rollup *could* look like in 2022
- MIT Bitcoin Expo committee 2021-2023
- Alpen Labs since April 2024, building **our Bitcoin rollup**
- I also really like Rust, we use it for almost everything!

Problem Statement

A Lightning channel is *(kinda)* a ~~blockchain~~ distributed state machine

- State
  - account balances for all (2) parties
  - set of HTLCs
  - whatever other magic tricks (discreet log contracts?)
- Operations (“Transactions”)
  - (Classically: Simple peer send)
  - Create HTLC
  - Destroy HTLC (preimage, timeout)
- Consensus: all (2) parties sign each new block (state)
- Finality: all (2) parties sign revocations for old state

Ok but really

- All parties need to be online ~concurrently
  - Infeasible to have more than 2 parties, leads to liquidity issues at scale
- All funds have some single owner, must be channel member
  - Hard to have DAOs, DEXes, etc
- Channels have to settle when any one party is dishonest
- Ethereum: Some direct generalizations were tried
  - State channels: more programmability
  - Plasma: more users

What we want

- A separate blockchain, running “on top of” Bitcoin (L1)
- Checkpoint state to Bitcoin periodically
  - ensure current state always *reconstructible* from Bitcoin data
- Rollup (L2) chain has ~native BTC, like Lightning
  - Minimal additional security assumptions
- Minimally restricted flow of funds between L1 and L2
- Rich computation!

Answer: (“zk” / validity) rollups!

Aside: SNARKs

- Succinct Non-interactive ARgument of Knowledge
- Attest to correctness of a statement, only revealing chosen variables

“f(w) = x, for some value of x”

- Cheaper to verify a snark of a program’s execution than to just execute the program
- Shifts trust from one place to another, or one blockchain to another
- “zk-SNARK”: also gives total privacy, but we don’t actually need that
- Various kinds: Groth16, PLONK, STARK, etc.

Ethereum has it easy

- Persistent contract storage
  - SSTORE, SLOAD
- EVM is highly expressive
  - JUMP, JUMPI, CALL, etc
  - Arbitrary math
  - Arbitrary cryptographic primitives, much more cheaply
- Easy data publishing
  - Very generous calldata limits, no need to inscription-like envelope hacks
  - Now they get data blob shards, even cheaper
- Directly control flow of funds by contract code

Breaking down the problem

1. Sign arbitrary data on Bitcoin for use in scripts

2. Verify “arbitrary statements” (snarks) with Bitcoin script

3. “Direct” flow of funds based on dynamic data (rollup state)

BitVM2

Quick primer on Lamport signatures (1 bit)

Keygen():

1. Sample random k_0, k_1
2. Output
   a. private key

      $k = (k_0, k_1)$

   b. public key

      $P = (H(k_0), H(k_1))$

Sign(k, b):

1. Output signature: k_b

Lamport verification in Bitcoin script (goal 1)

Stack Elements

```
// Opening this bit commitment to the value "1"
<0x47c31e611a3bd2f3a7a42207613046703fa27496>
<1>

```

Witness Script

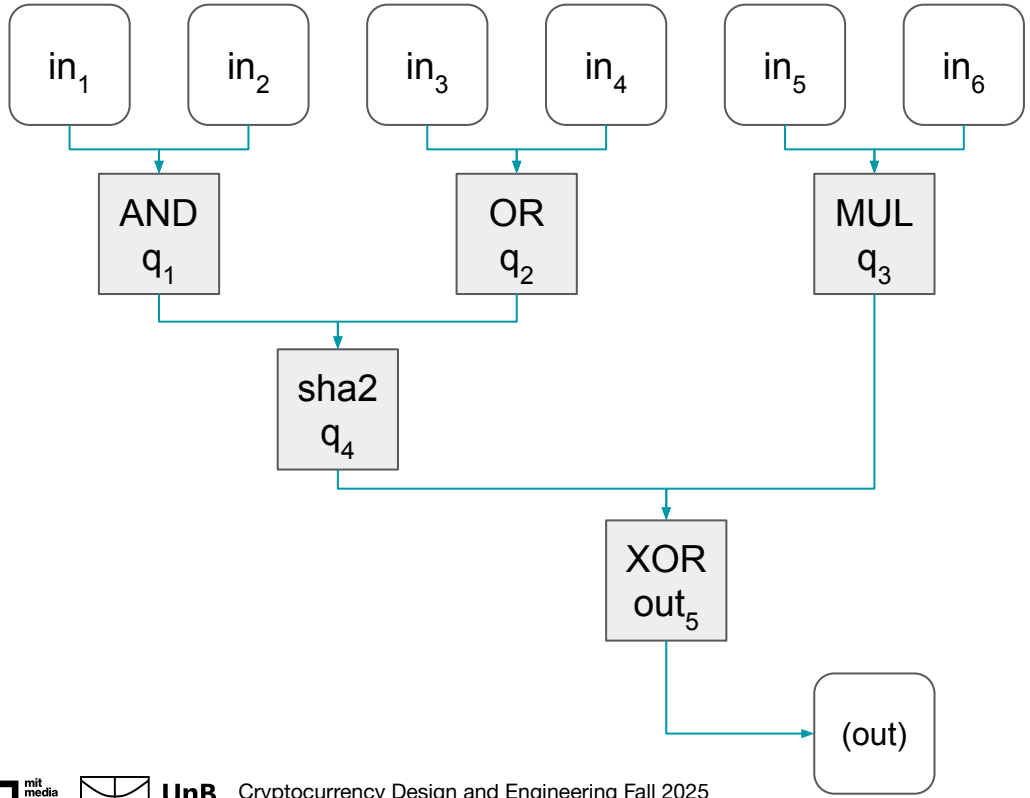
```
OP_IF
    OP_HASH160
    <0xf592e757267b7f307324f1e78b34472f8b6f46f3>
    OP_EQUALVERIFY
    <1>
OP_ELSE
    OP_HASH160
    <0xb157bee96d62f6855392b9920385a834c3113d9a>
    OP_EQUALVERIFY
    <0>
OP_ENDIF

// Now the bit value is on the stack
```


Source: Robin Linus, bitvm.org/bitvm.pdf

BitVM2 computation (goal 2)

- Control spend path from utxo on satisfaction of ~arbitrary circuit
  1. Prover publishes each wire value as Lamport-signed bits
  2. “Disprove” script tapleaf for each gate, spend proves assignment invalid
  3. On timeout, assume assignment valid, allow happy spend tapleaf
- Limited expressiveness from Bitcoin script
  - Have to reimplement addition/multiplication/hashing/etc gadgets
  - ~400 kvB tx size limit, 1000 element stack depth limit
- Doesn’t allow *data-dependent* spends, but that’s enough
- Developed by “BitVM alliance”: Robin Linus, Alpen, various others

Circuits



Tapleaf	Wires
1	in_1, in_2, q_1
2	in_3, in_4, q_2
3	in_5, in_6, q_3
4	q_1, q_2, q_4
5	q_3, q_4, out_5

NAND gate script (BitVM1)

```
// Reveal preimage of hash C1 or hash C0
<0xC468A29472CACF3EF179BA2352F88587B91E3E15>
<0x829923B22B9E831822E0A783F92687D27128157B>
OP_BITCOMMITMENT
// Now the bit value of "C" is on the stack
// ... we put it to the altstack for now
OP_TOALTSTACK

// Reveal preimage of hash B1 or hash B0
<0x34F0132278E874836DA82F8A6C1E10A21A153D17>
<0xF9FCE46CEFE9D9392108480AD42B4CE69557D27D>
OP_BITCOMMITMENT
// Now there's the bit value of "B" on the stack
// ... we put it to the altstack for now
OP_TOALTSTACK

// Reveal preimage of hash A0 or hash A1
<0x5ACFDE72A8E37111CBA96D3DD705BA983F47AF4D>
<0xA0172816A2D1B20EF0D5A1093958E9564E590BAF>
OP_BITCOMMITMENT
// Now the bit value of "A" is on the stack
```

```
//
// Verify that "A NAND B == C"
//

// Read "B" from altstack
OP_FROMALTSTACK

// Compute "A NAND B"
OP_NAND

// Read "C" from altstack
OP_FROMALTSTACK
// ... and check that it is correct
OP_EQUALVERIFY
```

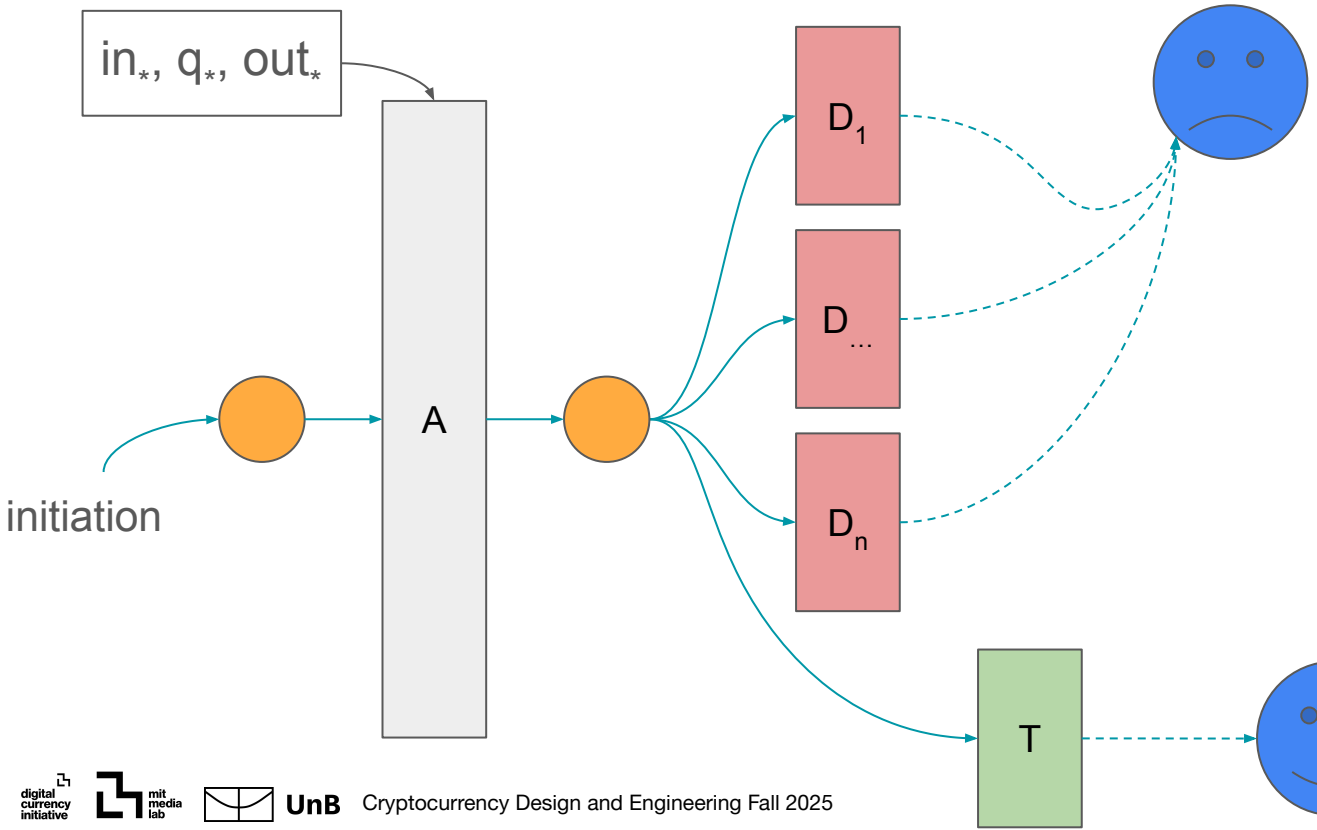
BitVM2: Invert this check

Source: Robin Linus, bitvm.org/bitvm.pdf

Fake covenants (goal 3)

- Constrain how future coins spent in a predefined tx graph
- Limited, but straightforward solution
  - Pick n parties as signers (like guarantors)
  - Sign n-of-n multisig for the whole tx graph
  - Share all the signatures with everyone
- For basic protocol security, we don’t have to think about it too hard
  - n-of-n multisig, so only any one party needs to be honest
  - Protocol economics is a little more complicated

BitVM2 transaction graph (also goal 3)



- Assert

then

- Disprove$_i$

or

- Timeout

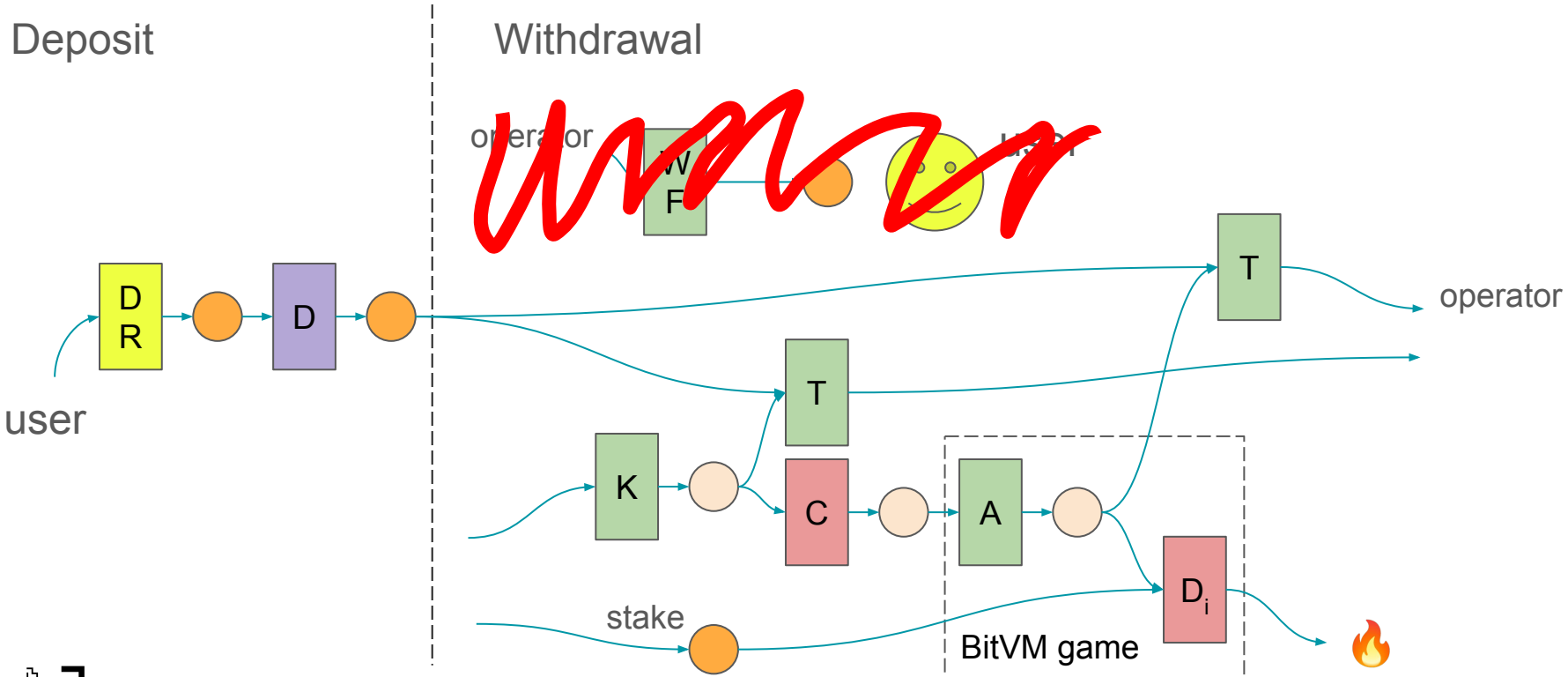
Realistically, what can a script gate do?

- Winternitz signatures instead of Lamport signatures
- Control flow is the same
- Integer addition is cheap
- ~4 254-bit prime field integer multiplications (bn254 curve field)
- One blake3/SHA-256 hash, some other stuff

Finally: Making a *snark-based* bridge

- We don’t have covenants, so perfect designs are just impossible
- *But we can do it optimistically!*
- New “operator” entity
  - Holds liquidity
  - Fulfills withdrawals, gets reimbursed if honest
  - Attest to a proof of honesty, BitVM2 slashes a stake if invalid

Hypothetical bridge transactions



Things I’m glossing over

- Combinatorial blowup of withdrawal game instances
- Rollup checkpointing and withdrawal issuance
- Precise particulars of the proof statement
- Stake bookkeeping txs (input to kickoff tx)
- Punishing malicious challenges (ie. griefing honest operator)
- Long-range attacks (ie. secret forked chain)

More detail

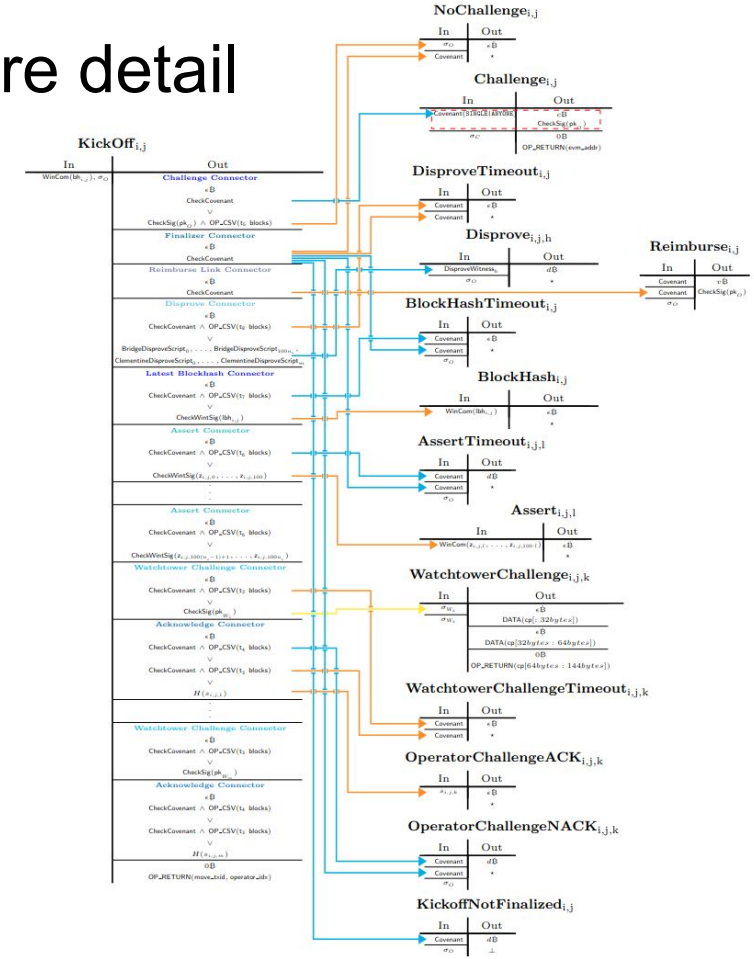


Fig. 2: **KickOff** transaction graph.

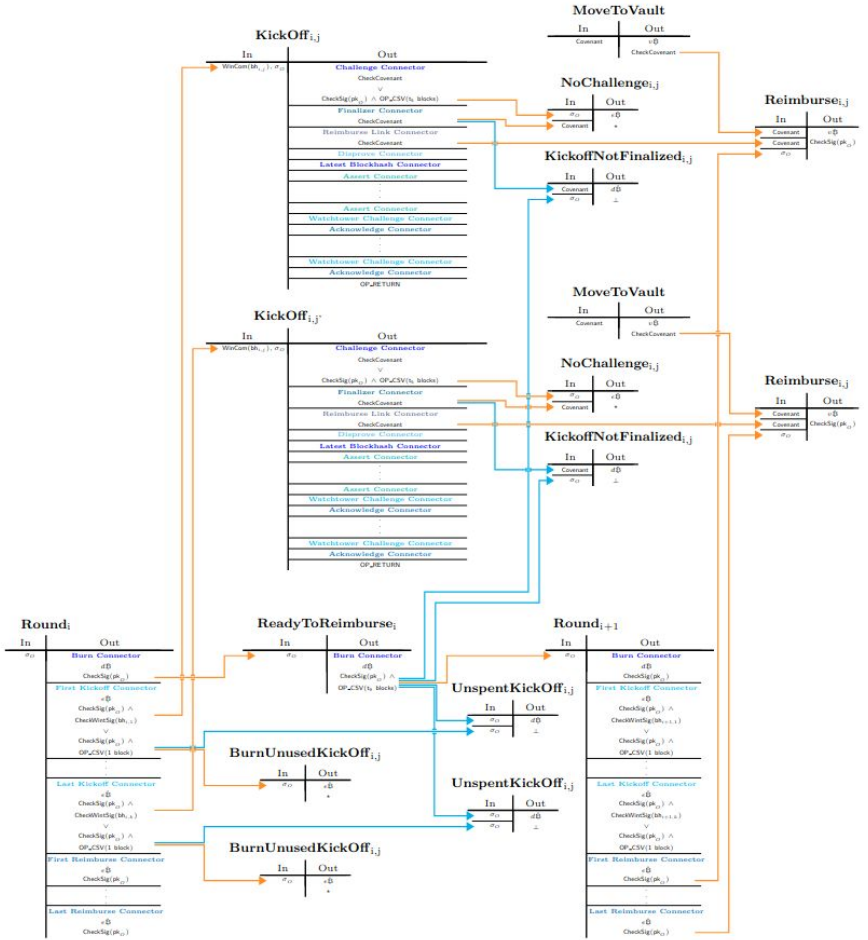


Fig. 4: Round transaction graph in the presence of multiple (in this case, due to space constraints, two) different **KickOff** transactions.

Source: Citrea, eprint.iacr.org/2025/776.pdf

Beyond BitVM2

BitVM2 is still pretty limited

- It's *really complicated* and *expensive*!
  - Hits tx size limit in many places (399kvB @ 50 sat/vB = 0.19 BTC)
  - Have to pre-commit to fee rates in several places, usually no CPFP
  - Complex script gadgets, ton of effort to audit
  - Packing puzzle to organize various script sizes/shapes

- Economics of deposit size and stake size are complicated
  - Have to carefully balance operator vs challenger costs
  - Stake and velocity impacts entire bridge liquidity and fee structure

BitVM3: Garbled circuits for Bitcoin, really fast

- GC: Single-use outsourced private computation
  - Cheaper variants without data privacy
- Similar Lamport-like commit-reveal
  - Sign only circuit input wires instead of all intermediate wires
- Circuit decrypts a secret if proof is invalid
  - Use the secret to trigger the disprove path, checks secret's hash
  - Unhappy case txs are *tiny*! (~440 vbytes)
- New issue: How to ensure correctness of the circuit and hashes used in the txs?

Glock!

Glock *(Garbled Lock)*

- Next-gen bridge protocol in BitVM3 family
- Similar tx structure to BitVM2 bridge
  - Remove BitVM2-style gate disprove scripts, just check secret reveal
  - Possible simplifications still being worked on
- Much more complicated off-chain signalling to agree on tables
- Dramatically changes stake economics
  - Maybe between 400x-1000x cheaper
  - Lower L1 fees, more operators, more liquidity, better user experience!
- **Unfortunately:** Still no *perfectly* unilateral exit as long as we need operators

Comparison

	BitVM1	**BitVM2**	**BitVM3 / Glock**
Verification	on-chain	on-chain	off-chain
Challenge rounds	O(#gates)	1	1
Assert size (total)	O(#wires)	O(#wires)	O(#inputs)
Gate check tx size	O(max(gate))	O(max(gate))	O(1)
Gate types	script	script	AND, XOR

Questions?