



# Getting Started with BitVMX

A Practical Approach

# What is BitVMX?

- 
- Disputable Computation on Bitcoin
  - Allows defining UTXO spend conditions based on program results
  - Any program compiled into a RISC-V binary
  - For example, a ZKP Groth16 verifier (but could be others)
  - If all agree off-chain → Happy path (99% of the time)
  - If not → On-chain dispute

# Applications Using BitVMX



## Union Bridge

- Decentralized bridge
- 1/n honest assumption
- Bridge BTC → RBTC (on the Rootstock L2 blockchain)

## Cardinal

- Bitcoin NFT/Ordinal DeFi mode
- 1/n honest assumption
- Allows to lock an ordinal and trade it on Cardano Blockchain
- Uses T.O.O.P to reduce the need for operator collateral

# Hackathon



Berlin | October 2-4, 2025



# BitVMX Track



The Lightning++ hackathon includes an optional BitVMX challenge with a **2,000,000 sats** prize!

Judges will primarily use the presentations to select winners with the following criteria:

- **Routine Difficulty:** Project idea, potential impact, ambition.
- **Routine Execution:** Achievement, what you actually built, does it work?.
- **General Effect:** Wow Factor, presentation, applicability to theme.

# What can I do?



BitVMX lets you design **optimistic protocols** on Bitcoin. Perform computation off-chain and use Bitcoin L1 only for fraud proofs & settlement. For example:

- Trust-minimized **cross-chain bridges**, BitVMX verifies bridging logic between Bitcoin and another chain (e.g. RSK, ETH rollup).
- Bitcoin-native **stablecoins** or decentralized **derivatives**, BitVMX manages collateral, liquidation logic, and enforces mint/redeem rules.
- Verifiable **games** where moves occur off-chain but disputes are resolved on-chain.
- **Tournaments** and **betting** systems settling in BTC without trusted intermediaries.

## Example App



Check out our example repo <https://github.com/FairgateLabs/bitvmx-hackathon-games>



It includes:

- Visual frontend to help understand the concept
- BitVMX-powered backend
- Documentation

You can learn more at our [Knowledge Hub](#)

Or talk to us directly at [BitVMX Devs Telegram](#)



# Using BitVMX



# Communication



## Message Broker (using Tarpc)

- Sends messages between internal and external BitVMX components
- TLS Encrypted
- Pinned Certificates for Client and Server
- Allow list
- Routing

## Communication - Code Example

```
pub fn init_broker(role: &str) -> Result<DualChannel> {
    let config = Config::new(Some(format!("config/{}.yaml", role)))?;
    let broker_config = BrokerConfig::new(config.broker_port, None);
    let bitvmx_client = DualChannel::new(&broker_config, L2_ID);
    Ok(bitvmx_client)
}

let msg = IncomingBitVMXApiMessages::GetPubKey(funding_public_id, true)?;
bitvmx_client.send(BITVMX_ID, msg.to_string())?;
```

# BitVMX Client



- It's where the **Protocols** live, and where you need to add new ones.
- Uses BitVMX CPU to verify **Program's** execution through the BitVMX Job Dispatcher
- Connect with other **Operators** BitVMX Clients to setup the protocols
- Tracks and dispatch **TXs** from the protocols
- Automatically reacts to specific TXs
  - Collaborate with valid flux
  - Challenge malicious activity

## Setting Up Program and Protocol - Code Example

```
let program_id = Uuid::new_v4();  
let program_path = "../BitVMX-CPU/docker-riscv32/riscv32/build/hello-world.yaml";  
  
let msg = incomingBitVMXApiMessages::SetVar(program_id, "program_definition",  
VariableTypes::String(program_path.to_string()))?;  
bitvmx_client.send(BITVMX_ID, msg.to_string())?;  
  
let msg = IncomingBitVMXApiMessages::Setup(program_id,  
PROGRAM_TYPE_DRP.to_string(), vec![p2p_address_1, p2p_address_2], 1)?;  
bitvmx_client.send(BITVMX_ID, msg.to_string())?;
```

# BitVMX CPU

- Maps every RISC-V instruction into Bitcoin Script
- Off-chain evaluation of the **Program**
- Dispute logic to identify the faulty instruction in case of a challenge
- The CPU runs as a separate process inside [BitVMX Job Dispatcher](#) to allow potential multiple parallel executions.



## Program - Code Example



```
#include <stdint.h>
#include "emulator.h"

int main(int x)
{
    unsigned int *a = (unsigned *)INPUT_ADDRESS;
    unsigned *b = a + 1;
    unsigned *c = a + 2;
    if (*a + *b == *c) { return 0;}
    else {return 1;}
}
```

## Program - Build

- We need to build our program into **RISCV32** compatible **.elf** files
- Repo with helpers to build the program

<https://github.com/FairgateLabs/bitvmx-docker-riscv32>

- build the images run the corresponding script for Win,Linux,Mac (docker-build.bat, docker-build.sh or docker-build-mac.sh )
- Compile a program example `docker-run.bat riscv32 riscv32/build.sh src/hello-world.c --with-mul`
- Compile a ZKP groth16 verifier `docker-run.bat verifier verifier/build.sh --with-mul`

# Program - Yaml



```
elf: add-test.elf
nary_search: 8
max_steps: 50
input_section_name: .input
inputs:
  - size: 8
    owner: const
  - size: 4
    owner: prove
```



Execute Order 66

VIA 9GAG.COM



You want fries with that?

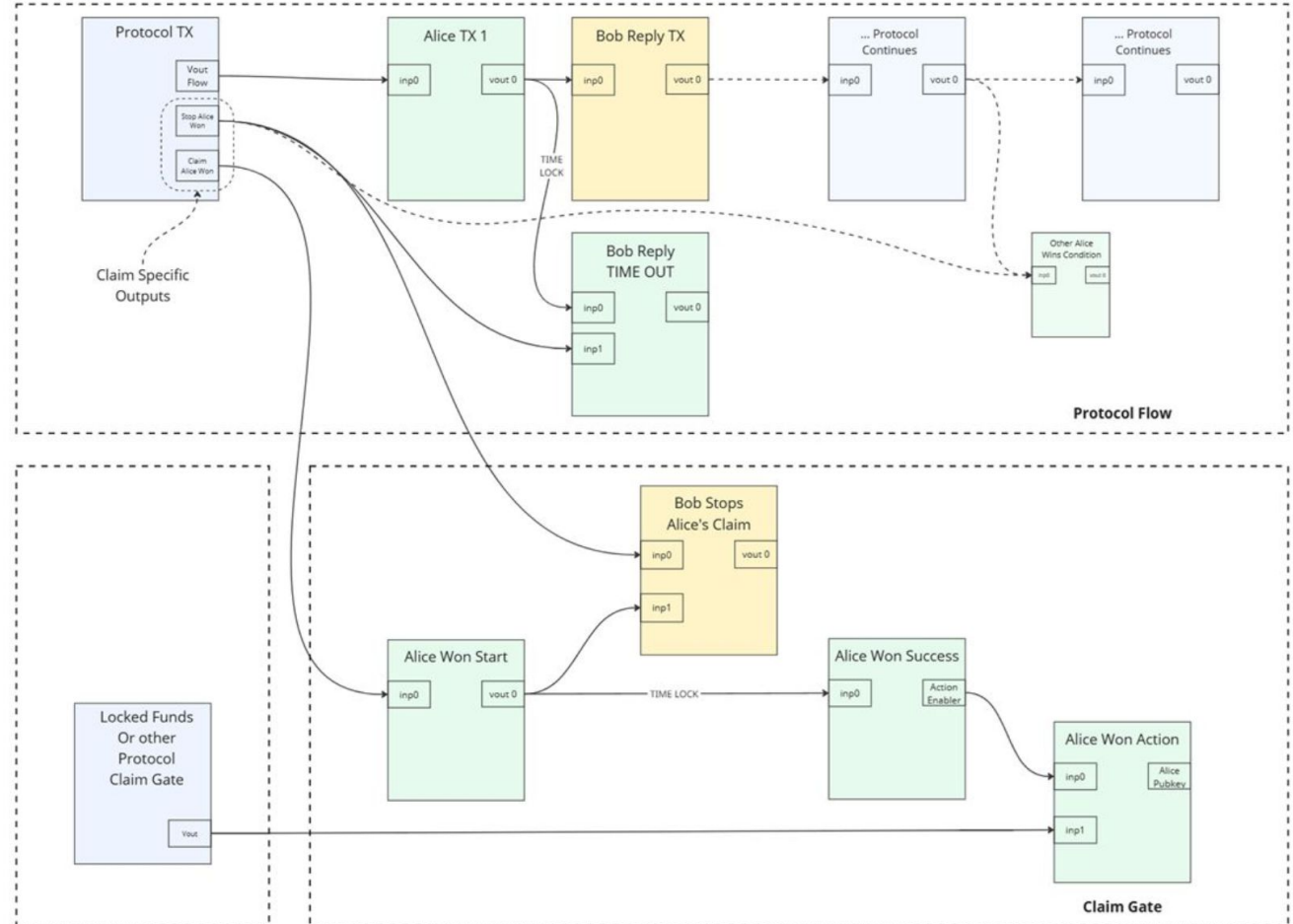
# BitVMX Client - Protocols



Protocols are like plugins for BitVMX, adding extra functionality. They can:

- Set flows using Directed Acyclic Graphs (**DAGs**) of pre-signed Bitcoin transactions
- Store and read information
- Send transactions and receive updates about them
- Sign with aggregate keys (MuSig2 multisig)
- Sign using Winternitz keys
- Hold the program definition

# Protocols



# Creating a Protocol - Protocol Handler



To add a protocol in BitVMX we need to create it at [src/program/protocols](#) and implement [ProtocolHandler](#), where you can add functionality like:

- Generate keys
- Define Protocol Setup
- Receive transactions news
- Trigger transactions on finish

# BitVMX Protocol Builder

Creates complex protocols DAGs using Bitcoin transactions

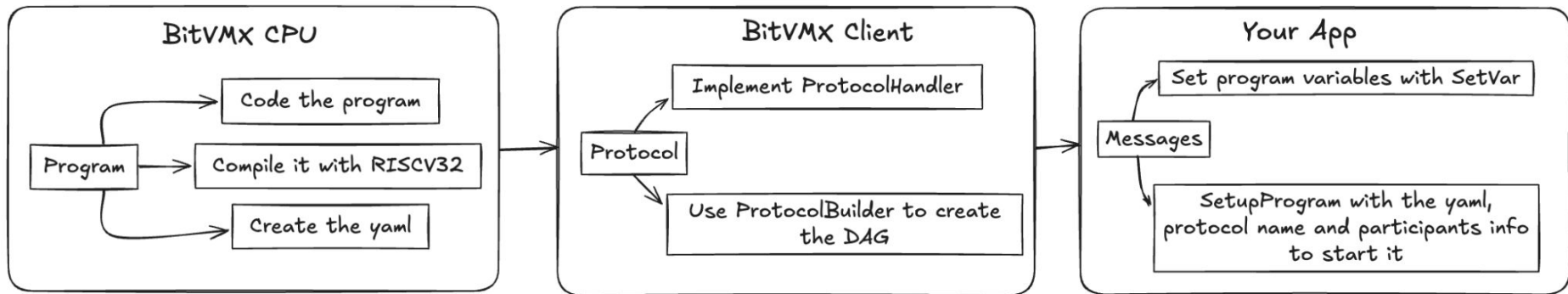
- Creates and stores different types of transactions (Segwit, Taproot)
- Stores metadata to know the inputs, outputs, spending paths, and signatures
- Speed-ups using Child-Pays-for-Parent (CPFP) transactions
- Dust and TxFee calculation
- Graph visualization

## BitVMX Protocol Builder - Code example

```
// Add the output to the claim transaction that contains two leaves:
// 1. The aggregated signature for the stoppers
// 2. The timelock script that will be used by the claimer if he succeeds the claim
let verify_aggregated = scripts::check_aggregated_signature(&aggregated,
SignMode::Aggregate);
let timeout = scripts::timelock(timelock_blocks, &aggregated, SignMode::Aggregate);

let start_tx_output = OutputType::taproot(
    amount_fee + ((1 + actions.len() as u64) * amount_dust),
    aggregated,
    &vec![verify_aggregated.clone(), timeout],
)?;
protocol.add_transaction_output(&stx, &start_tx_output)?;
```

# Summary



**MAY THE**  
**BITVMX**  
**FORCE**  
**BE WITH YOU**

# Thank You!



<https://bitvmx.org/>



<https://www.fairgate.io/>



<https://github.com/FairgateLabs>



BitVMX  
**FORCE**

A community-supported initiative  
to drive the establishment of  
**BitVMX** as the solution of choice in  
Disputable Computing on Bitcoin

October 2025