BitSNARK & Grail

Bitcoin Rails for Unlimited Smart Contracts & Scalability

Ariel Futoransky, Yago and Gadi Guy

April, 2024

Abstract

In the December 2023 paper, *BitVM: Compute Anything on Bitcoin*, Robin Linus introduced a new computing paradigm for Bitcoin, allowing the expression of Turing-complete smart contracts. The new paradigm generated significant excitement focused on a narrow but profound purpose – the ability to develop near trustless Rollup bridging for Bitcoin, without any changes to Bitcoin's consensus rules. Despite introducing the possibility of Rollup bridging, BitVM, as a general purpose system, is not optimized for this purpose. We present *BitSNARK*, an optimized system for the development of Bitcoin Rollup Bridges. *BitSNARK* builds on the BitVM paradigm to improve, by an order of magnitude, program size and reduce challenge/response length. Additionally, we present *Grail*, an implementation of *BitSNARK*, creating a practical and scalable Bitcoin Rollup Bridge.

Introduction

In the beginning, Satoshi Nakamoto created Bitcoin. Ever since debate has raged on how best to both scale Bitcoin and introduce new functionality. The debate has remained almost unchanged in its fundamentals since Bitcoin's inception. At its heart is a seemingly intractable trade off between three desirable features: scale, computational expressivity and decentralization. Improvements to one feature causes deterioration in the other two features. Thus the debate around this trilemma has raged on, to all appearances, without possible resolution.

In 2013, Bitcoin Core developer Gregory Maxwell, introduced a possible solution to the trilemma. With "bleeding-edge cryptography" it would be possible to achieve the holy grail of scale, Turing-complete programmability while still maintaining decentralization. The cutting edge cryptography described was SNARKs and would allow the creation of special environments, in which any arbitrary program could run. The transactions executed in such environments would create "quickly-checkable proofs" that would retain Bitcoin security. However, at the time it was believed that introducing the ability to verify SNARKs would require a softfork and this tantalizing solution to the trilemma has remained only a thought-experiment.

Ten years after Maxwell first presented the solution offered by SNARKs, many had come to believe that no solution to the trilemma was practicable and that Bitcoin would forever remain with the limitations of its genesis. Indeed, the cognitive dissonance led many in the community to make peace with the trilemma by denying that scalability and expressive programmability were desirable at all. Despite this, the verdict that the trilemma was unsolvable would prove premature. In 2023, Robin Linus introduced *BitVM*, a new computing paradigm for Bitcoin. While BitVM introduced the possibility of running any program on Bitcoin, its most promising revelation was the possibility to verify SNARKs, allowing the creation of Bitcoin Rollup Bridges. These bridges would allow Bitcoin assets to be used in special environments,

called rollups. These Rollup environments could offer both scale and the ability to compute anything. Finally, a workable solution to the Trilemma was at hand.

However, due to the general-purpose nature of BitVM, many significant limitations have come to light making development of a practical, production-ready bridge difficult and complex. One limitation is the need for pregenerating, presigning and storing billions of transactions, one for each potential challenge in a verification game. This complexity has called into question the practical applicability of BitVM for Rollup Bridges without significant tradeoffs.

In this paper, we build on BitVM to present two innovations allowing practical Bitcoin Rollup Bridges with minimal tradeoffs. The first innovation is *BitSNARK*. *BitSNARK* is a software library that allows verifying SNARK proofs on the Bitcoin blockchain, by way of an interactive protocol between two or more parties. *BitSNARK*, instead of a general purpose system, is an application specific system optimized for SNARK verification. It offers an order of magnitude improvement to program size and as much as 50% reduction in challenge/response protocol length for the kind of computations required by bridges. Further, only a single challenge type targeting equivocations is required, where BitVM requires 6 different challenge types. Additionally, our simplified protocol increases the security, transparency and auditability of the system.

The second innovation we introduce is *Grail*. *Grail* comprises 3 main components: (1) *BitSNARK*, for the verification of SNARKs. (2) A prover system for the creation of SNARK proofs for both Bitcoin transactions and Rollup transactions. (3) The Grail Bridge, an asset bridge that securely transfers assets between Bitcoin mainchain (Layer 1) and Rollups (Layer 2).

BitSNARK

Several approaches have been suggested to the problem of running general purpose programs on the Bitcoin blockchain. Due to the extremely limited nature of Bitcoin script, these approaches require either slow and expensive on-chain computation, or vast off-chain computation. *BitSNARK* focuses on the most useful use-case for this technology – zkSNARK verification – and offers a much faster and less costly solution by implementing a highly optimized SNARK verifier designed to run on the Bitcoin blockchain without requiring complicated general purpose computation.

In principle other algorithms can be compiled as a *BitSNARK* program, but the main use-case for *BitSNARK* is the execution of a zkSNARK verification program pre-configured with a verification key. The program takes as input a few hundred bytes of a SNARK proof and either accepts or rejects it based on the calculation associated.

BitSNARK VM

BitSNARK VM is a virtual machine, emulating a simplified register-based processor with only three instructions, natively supporting the finite field calculations required for elliptic curve pairing operations. As a result, the verification protocol is notably simplified, needs no memory consistency checks, and requires only two challenge scenarios: single-instruction execution error proofs and reveal-equivocation proofs.

The VM has a limited number of 256 bit registers, each with a unique ID. Each instruction receives register IDs, performs a single calculation and emits its result into the target register. Certain registers can be marked as immutable, so their values cannot be modified and they can be optimized in the Bitcoin script. The following instructions are supported:

- addmod(t, a, b, m) – add the values of registers a and b, modulo c, into register t.
- andbit(t, a, b, c) – if bit b of register a is 1, write the value of register c into register t, otherwise write the value 0.

- equal(t, a, b) – if the values of registers a and b are equal, write 0 into register t, otherwise write 0.

Additionally, an attempt to write a value into an immutable register results in the program being rejected if the value being written is different from the value in that register.

It can be demonstrated that these instructions are sufficient to implement a zkSNARK verifier.

Interactive Verification Protocol

BitSNARK is designed as a two-party protocol for a prover and a verifier, where the prover initiates the execution by revealing the program's input and its result, and the verifier can in turn dispute it if they believe the claim is incorrect. When considering more than two operators, a two-party *BitSNARK* protocol is set up for each pair of agents allowing any successful two-party challenge to block an invalid program execution.

The protocol is organized as a series of challenge and response interactions. The prover creates a transaction revealing the required proof as input. It also engages with the verifier via a peer-to-peer protocol to create a set of pre-signed transactions. These transactions can be used by the two parties to perform the steps of the protocol.

During each step, the prover cuts the program execution in half, and commits to the state of the virtual machine at that point of incision. The verifier chooses which of the two resulting parts they believe is false. This continues until the prover has committed to a single *BitSNARK* operation. This operation can then be executed on-chain, in order to determine the winner of the protocol.

If no challenge is entered during the allowed time or if the verifier fails to demonstrate a rejection, the funds are unlocked and the prover can make use of the funds. On the other hand, if the challenge is successful, the funds remain securely locked until any other operator initiates the withdrawal process on their own. The verifier is incentivized by receiving a sum from an output created beforehand by the prover in the initiating transaction.

Each step of the interactive protocol is a time-locked transaction, such that if a party walks away, they lose the game once that timeout has expired.

The complexity of the protocol is $O(\log(n))$, which allows the program being executed to be very large while efficiently scaling the number of steps, keeping them small.

Security Assumptions

While many systems rely on a majority vote in a threshold signature scheme for security, *BitSNARK* promises to provide stronger security by allowing a single honest agent to prevent abuse by any or all of the other agents.

An initiating agent is required to create a transaction containing an output with some minimum amount of Bitcoin, which is forfeit to any verifier who successfully proves that the transaction is fraudulent. This in turn incentivizes agents to keep track of blockchain transactions in order to find opportunities to engage in the verification protocol. The verifier is also required to attach an output to his challenges in order to penalize verifiers for making challenges in non-fraudulent cases.

The result of this mutual incentive scheme is that the cost of engaging in the protocol does not fall on the user of the system; instead – it is covered by the dishonest participant. This "optimistic" approach allows us to keep costs down to a minimum.

Grail Bridge

The bridge is intended to allow users to transfer assets between the Bitcoin blockchain and an L2 (Layer 2) network.

To achieve this, the bridge requires agents, here called operators. Operators are responsible for monitoring the bridge operations and assisting users with deposit and withdrawals. They are required to constantly monitor both the Bitcoin as well as the rollup networks, and participate in the defined protocol steps. In addition, operators are required to register and maintain an active internet endpoint to support P2P interactions with the other operators.

A minimum of two operators are required to keep the protocol going, but any number can be supported. Grail allows changes to the operators group: new operators can request to join or leave by following the required steps.

Operators can lock funds on the Bitcoin side by sending them to Taproot addresses created using *BitSNARK*. The funds are thereby locked in a UTXO until a SNARK proof is provided that allows them to be retrieved. On the L2 side, the operator sends a SNARK proof to the bridge smart contract, thereby causing the bridge to mint tokens to the operator's wallet.

In the reverse process, an operator burns their tokens via the smart contract, thereby obtaining a SNARK proof that allows the operator to retrieve their funds, on the Bitcoin side, from the UTXO, using the *BitSNARK* protocol.

Operator Registration and Resignation

To become an operator, an agent first creates a public key identity and an exit secret, and uses the bridge's smart contract to register both his public key and the hash of the exit secret. Afterwards he deposits the required stake, which gets locked as guarantee while the operator is in function.

The stake can be recovered when the operator requests to exit the protocol role, provided that no violation has been demonstrated to that point. To claim the stake and exit the role, the operator reveals the exit secret, which can be used in turn to forbid them from participating in any future protocol exchange.

The active group of operators maintains a multi-signature identity that is used to pre-sign transactions and optimize operational cost on non-adversarial scenarios. The bridge uses the MuSig2 two-round protocol to create Schnorr signatures approved by the full set of operators. In steps requiring pre-signed transactions, the group identity is used to enforce the possible protocol state transitions. Pre-signs are required, for example, to realize stake funding, as well as bridge deposits.

In addition, some optional optimization functions can be performed by having direct approval of the entire operator group. For example, simplified direct withdrawals or UTXO consolidation.

Grail Bridge Deposit

In order to efficiently perform new deposits on the bridge, three smart contract functions are used to coordinate the operations.

- Initiate deposit – A user specifies a set of funding UTXOs on the Bitcoin network, and requests an address to make a deposit. The operators work together to generate an address, and a set of presigned transactions.
- Confirm deposit – As soon as a set of operators have signed the required transactions, the deposit is confirmed and the user can initiate the transfer. The user can verify that the address has been generated correctly and that all the requisite transactions have been signed before transferring their funds.

- Complete deposit – Given a proof of transfer to the accepted bridge deposit address on the bitcoin network, the equivalent amount of the rollup's ERC20 token is transferred to the user in the rollup. The locked UTXOs are registered in the smart contract.

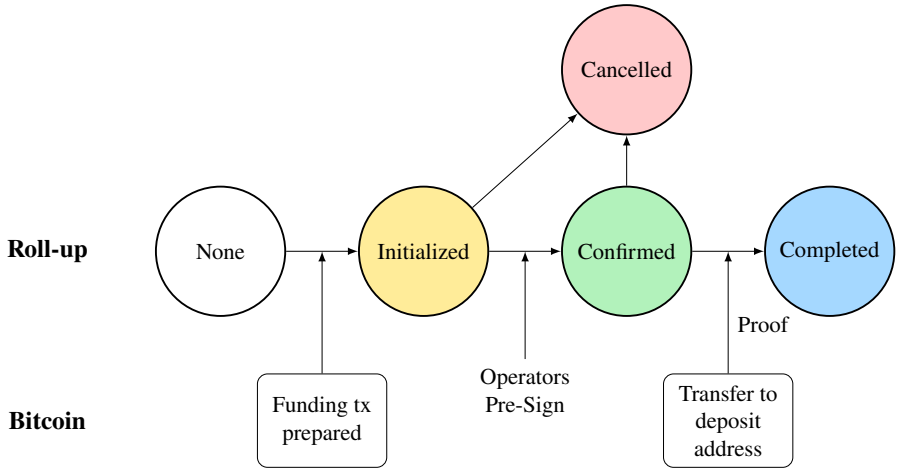

Figure 1

Operator-initiated Withdrawal

In an operator-initiated withdrawal operation, an operator holding tokens will exchange them for some of the UTXOs locked on the bridge.

To complete that function, the following smart contract method is used:

- Request withdrawal
  The operator requests a withdrawal and specifies an amount and target address. The smart-contract collects the tokens equivalent, selects a group of UTXOs that adds to the amount, and assigns those UTXOs the the operator address.

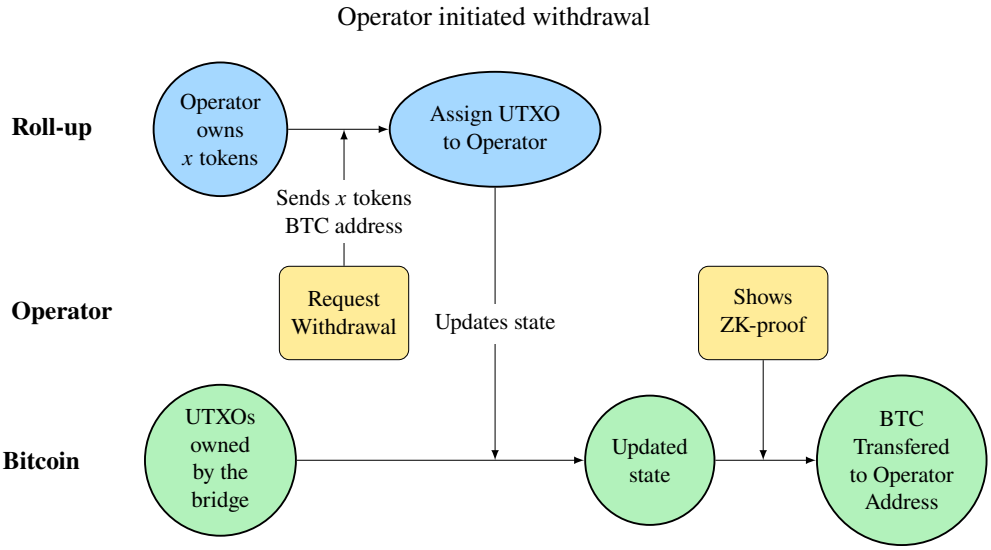

Figure 2

Afterwards, when the smart-contract state has been committed to the bitcoin network, the operator can use a zk-proof showing that the UTXOs has been assigned to him to transfer the funds in the bitcoin network.

Coordinated withdrawal

In the event all operators are available and cooperating, an operator-initiated withdrawal can be completed by having a group signature of the transfer to the target address, avoiding the need to calculate and commit the zk-proof.

Operator-assisted Withdrawal

If a user requests to withdraw tokens, the operation can be completed by an operator holding Bitcoin. The bridge smart contract helps in performing a swap, bitcoin in exchange for tokens. This helps to minimize wait times and simplify the user experience. In exchange the operator gets a fee.

The following smart contract functions are available:

- Initiate withdrawal – the user specifies an amount and target address. The smart contract collects the funds and registers the withdrawal request.
- Operator assignment – an operator responds and selects a set of UTXOs they control to service the request. The withdrawal gets assigned to them. As soon as the state is confirmed, they transfer Bitcoin to the target address using the preselected UTXOs.
- Complete withdrawal – the operator completes the operation by presenting a proof of inclusion in the Bitcoin blockchain of the transaction to the target address. The smart contract transfers the target amount in tokens to the operator.

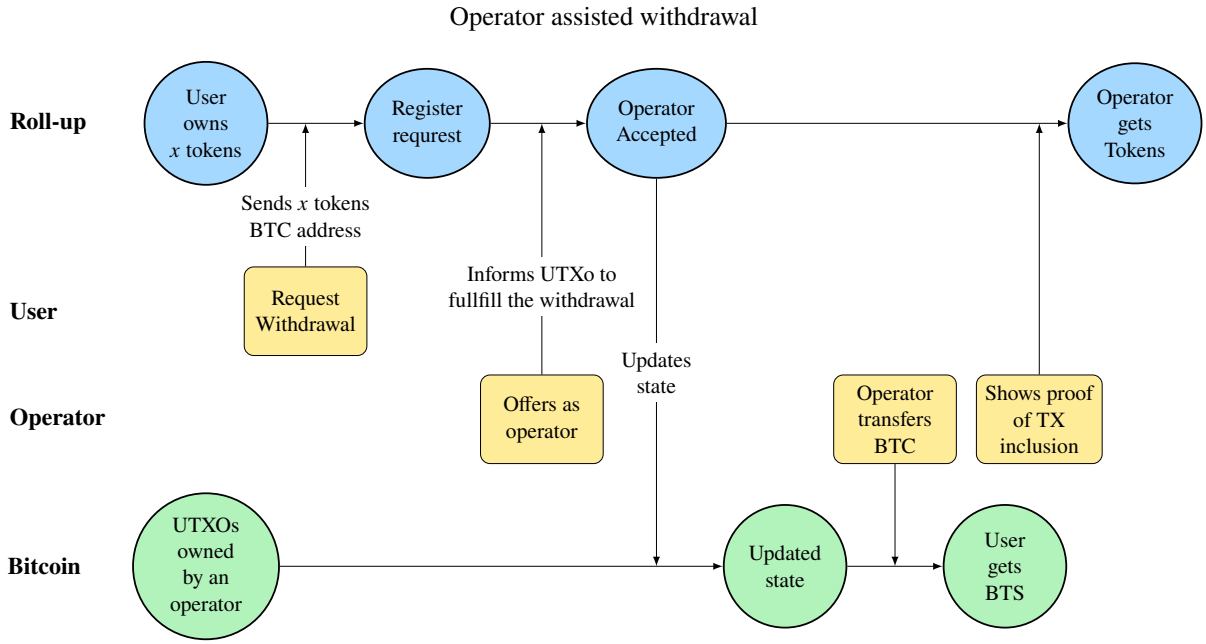

Figure 3

Proofs

Two types of proofs are used in the Grail bridge to support its operations: proof of transaction inclusion on the Bitcoin network, and proof of state transition of the bridge smart contract on the L2 side.

For both types, a zkSNARK groth16 proof is generated and verified. Proof recursion is expected to be supported, allowing proofs from other systems, such as Plony2 and STARKs.

Proof of transaction inclusion on the bitcoin network – Given a transaction hash, a proof that the transaction has been mined on the bitcoin network is generated by combining the following conditions:

- Proof that the transaction was included in a particular block
- Proof that a predefined number of blocks were confirmed afterwards
- Proof that a known accepted block is present in the chain

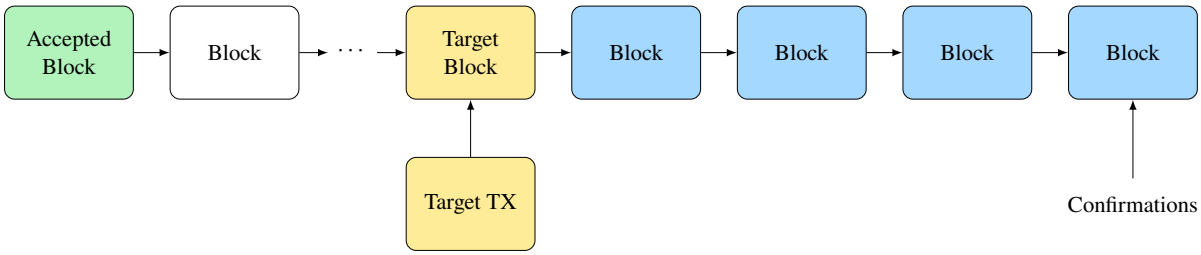

Figure 4

Proof that a variable on the bridge smart contract reached a particular state – Proofs of the bridge smart contract state are used in two cases: in order to confirm that a UTXO has been assigned to an operator for withdrawal, or to confirm that a stake has been released after an operator has resigned.

Depending on the design and security assumptions of the rollup the proofs can be constructed as:

- A proof of transaction inclusion on the bitcoin network – provided that roll-up is committing transaction information to the bitcoin network as a data availability layer, and that the rollup has a dispute mechanism to reject invalid state transitions.
- A proof of storage state on the rollup's smart contract – an alternative is to show that a target variable with its value has been included in the smart-contract storage when calculating the rollup's block hash.

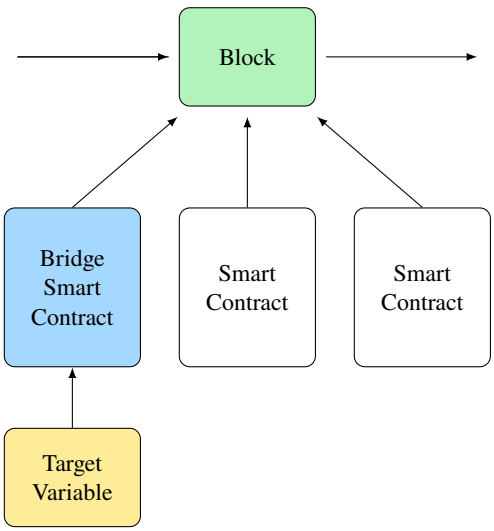

Figure 5

Advantages of the *BitSNARK* Optimized System

The main distinction between *BitSNARK* and BitVM is its processor architecture. Where BitVM uses a general purpose processor to allow the verification of a wide range of potential programs, *BitSNARK* uses a minimalistic design specifically targeting ZK-SNARK proof verification. As a result *BitSNARK* offer a simpler design with fewer components and fewer challenge scenarios required. This reduction in complexity directly translates to the trustworthiness, auditability and security of the *BitSNARK*.

Platform complexity

RISC-V 32 with base integer (RV32I) and multiplication extension (RV32M) has 55 instructions. Each instruction must be programmed in Bitcoin script in order to implement the BitVM protocol verification. In contrast, *BitSNARK* only has 3 instructions: ADD, SUB and MULBIT. *BitSNARK* doesn't require and doesn't implement any instructions for branching, conditional branching, looping or subroutine instructions.

RV32I has 4GB of addressable RAM. BitVM consequently needs to implement memory consistency in its protocol. BitVM does this by hashing the memory contents in each cycle and implementing a specific multi-step challenge response game to validate this computation. Thus, *BitSNARK* programs don't require RAM, and have no RAM consistency requirement at all.

Program Types

While BitVM can verify any program compiled into a RISC-V 32 binary, *BitSNARK* can only run ZK-SNARK proof verification programs. This apparent limitation can be overcome by using SNARK tools that allow proof of general purpose programs. For example the RISC ZERO platform allows ZK-STARK proofs of RISC-V execution, and later generating a groth16 verifier of the result.

Furthermore, for many applications, it will make more sense to have the protocol verify a SNARK instead of the whole computation. Firstly, this can reduce the total amount of on-chain computation. Secondly, a SNARK can significantly minimize the amount of input data required to perform the validation. Considering data availability requirements, this could have significant cost implications in many scenarios. One of these scenarios is bridging, for which *BitSNARK* optimizes.

There may be applications where directly using a general purpose platform is a preferred option to generating a SNARK (for example if some of the calculations performed limit ZK-recursion options).

Program Verification Protocol

BitSNARK challenge response is a straightforward two phase protocol, with only 1 challenge type. In contrast, BitVM is a more complex system, and as a result the challenge response game incorporates 6 different scenarios.

Technical Comparison

Feature	*BitSNARK*	BitVM
	Platform	
Registers	Variable number of 256-bit registers	31 32-bit registers
Instructions	3 add/sub/mulbit	55 47 from base/integer + 8 from multiplication extensions.
Random Access Memory	No	Yes Needs to implement memory consistency
Branching	No	Yes
Loops	No	Yes
Subroutines	No	Yes
	Program	
Program range	Zk-snark verification program	Any RISCV rv-32im program
Tool-chains	Libsnark, gnark, circom Any zk-snark generator library targeting groth16	LLVM / GCC Any compiler targeting rv32 bare
	Challenge Protocol	
Challenge types	1	6
Program loading	Fixed at setup	? Preloaded in memory. RO region
Input size	Approx. 350 bytes	Variable
Input signing	Lamport	Lamport / Winternitz

Critique of BitVM

On April 5'th, 2024, Tyler Whittle & Rijndael, published a critique of BitVM bridges: *BitVM Bridges Considered Unsafe*. In the essay, the authors argued that circumstances could arise where the funds seeking to exit the Bridge could exceed the capacity to redeem in a timely manner, causing catastrophic loss of funds. In future publication, we plan to provide a full analysis of Whittle and Rijndael's claims and demonstrate how Grail avoids such a circumstance. Here, we address the claims in abbreviated form.

Contra the system described by Whittle and Rijndael, Grail provides users a consistent ability to withdraw. This is achieved in the incentive aligned scenario (happy path) through optimistic consensus, which allows any amount of funds within the Grail bridge to be withdrawn to one or multiple addresses of withdrawing users. Taken to the limit, this would allow for the full amount of funds held within the Grail bridge to be redeemed within a single transaction, regardless of the value of funds held.

In the incentive unaligned scenario, the application of a fee-market for priority withdrawals coupled with the ability to scale operators via dynamic membership, allows for continued smooth operation of withdrawal. At all times, funds held in *Grail* are securely maintained with 1-of-n assurance. Any increase in liquidity requirements for withdrawal is accompanied with a complimentary increase in withdrawal fees. Operators have the ability to forward withdrawals to users making use of their own or borrowed funds. The rise in fees for priority withdrawals provides economic incentive for additional liquidity provision by either the existing set of operators or by attracting additional operators. Due to the fact that funds held in the bridge remain at all times redeemable by operators, any fees earned are effectively risk-free. The practical upshot is that Grail withdrawal fees instantiate a risk-free rate of return for Bitcoin.

Conclusion

In summary, *BitSNARK* and Grail build upon the monumental innovation of BitVM, forging a path toward practical, secure, and scalable Bitcoin Rollup Bridges. By optimizing for the specific requirements of SNARK verification, *BitSNARK* offers dramatically improved efficiency and reduced complexity compared to BitVM's general-purpose design. The Grail implementation leverages these advancements to create a complete system for Bitcoin Rollup Bridges that upholds the security and decentralization promises of BitVM while addressing the concerns raised by critics. Grail marks a significant step forward in resolving Bitcoin's long-standing scaling trilemma, paving the way for a future in which Bitcoin can fulfill its potential as the foundation of a new financial paradigm. Let's Build!

References

[1] Bitcoin Forum. *BitVM Bridges Considered Unsafe*. 2013. URL: https://bitcointalk.org/index.php?topic=277389.0.

[2] Ethereum Research. *Optimistic rollups*. 2022. URL: https://ethereum.org/en/developers/docs/scaling/optimistic-rollups/.

[3] Nakamoto, Satoshi. *Bitcoin: A peer-to-peer electronic cash system*. 2008. URL: https://www.ussc.gov/sites/default/files/pdf/training/annual-national-training-seminar/2018/Emerging_Tech_Bitcoin_Crypto.pdf.

[4] Robin Linus. *BitVM: Compute Anything on Bitcoin*. 2023. URL: https://bitvm.org/bitvm.pdf.

[5] Salvatore Ingala. *Merkleize all the things*. 2022. URL: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2022-November/021182.html.

[6] Tyler Whittle & Rijndael. *Really Really ultimate blockchain compression: CoinWitness*. 2024. URL: https://medium.com/@twhittle/bitvm-bridges-considered-unsafe-9e1ce75c8176.