Introduction

Hi all!

I'm Harry Kalodner, a Ph.D. student at Princeton and co-founder ofOffchain Labs. We're building a new layer 2 system on Ethereum for increasing the scalability and privacy of smart contracts based on our academic work Arbitrum, published in USENIX Security earlier this year.

We're currently working on building our deployment of Arbitrum on Ethereum. If you're interested in joining the effort, contact us at jobs@offchainlabs.com. For more info on Arbitrum, keep reading.

Arbitrum enables the creation of Arbitrum Virtual Machines (VMs) which are smart contracts: first class actors that can send and receive funds and messages as well as perform calculations and store data offline according to their code. Arbitrum VMs are much more scalable and private than conventional ways of implementing smart contracts.

Arbitrum does almost all of the management of VMs off-chain, with a minimum of on-chain work to ensure correct execution. To achieve this, when someone creates an Arbitrum VM, they select a set of managers who will be responsible for executing the VM. Arbitrum guarantees that any single honest manager can ensure that the VM executes correctly (even if all of the other managers conspire to try to cheat). And because hardly any data about a VM is ever put on-chain, Arbitrum VMs are much more private than many other system.

In particular, only a cryptographic hash of the VM's state is stored on-chain.

Arbitrum gives the managers of a VM incentives to agree on what the VM will do, and if the managers agree unanimously on what the VM will do, the system accepts that and it is recorded on-chain.

Dispute Protocol

If, contrary to incentives, the managers cannot agree on a VM's execution, they can fall back on an assertion/challenge/dispute protocol that's still very efficient. First, one manager makes a disputable assertion–a claim about what a VM will do. The asserting manager escrows a deposit as a guarantee that they're not lying. Then other managers have a fixed time period to challenge the assertion if they think it's wrong. If nobody challenges, the system accepts the assertion.

If somebody does issue a challenge, the challenger escrows a deposit as a guarantee that they're not lying. Now the system will referee a dispute protocol to determine who is lying, and the liar will lose their deposit. This happens in two phases:

The first phase relies on recursive bisection (based on Refereed Delegation of Computation, invented by Ran Canetti et al in 2011, and as used in TrueBit). The dispute protocol first demands that the asserter break their assertion into two half-size pieces; if the initial assertion covered N instructions executed by the VM, then each half should be an assertion about executing N/2 instructions. Next, the challenger must pick one of the two halves to challenge. At this point the size of the dispute (the number of instructions) has been cut in half. The same procedure is used recursively, until there is an assertion about a single instruction of execution that has been challenged.

The second phase requires the asserter to provide a "one-step proof" which demonstrates that starting with the initial VM state hash, executing one instruction will lead to the claimed final state hash. The cost of a one-step proof, in terms of size and proof-checking time, is a critical factor controlling the cost of dispute resolution. So we carefully engineered the Arbitrum VM architecture to make one-step proofs cheap. As an alternative, we could have used something like WASM, but that would have lead to substantially higher costs.

VM Architecture

Our academic paper goes into detail about the VM architecture. The bottom line is that

using our architecture, the total cost of a challenge including the bisection is O(log N + log log M)

. This is compared to standard architectures like WASM where the cost would be O(log N + log M + log C)

where N is the number of instructions executed, M is the size of the memory, and C is the size of the code. Arbitrum one-step-proofs are always under 500 bytes, and usually around 200 bytes.

Our implementation of Arbitrum on Ethereum will allow Arbitrum VMs to own and send any ERC20 or ERC721 token as well as Ether. VMs are also capable of receiving data from Ethereum smart contracts.

To provide a simple example of Arbitrum in practice, imagine Alice and Bob want to bet on a game of chess. Alice and Bob create a new VM which includes the rules of chess and select themselves as managers of the VM. Assuming they are honest and online, they can play the entire game off-chain in a typical state-channel-like fashion. Even in the worst case, either Alice or Bob can force the game to complete, either through timeout or victory, without having to execute any of the chess logic on chain. This allows the checkmate-checking logic to be simply encoded in the contract with guarantees of correct execution without ever having to execute the logic on chain.

Comparisons to Other Layer 2 Solutions

We believe Arbitrum holds some important advantages over other layer 2 systems.

Unlike Plasma, Arbitrum supports arbitrary smart contracts and doesn’t require long waiting periods for sending and receiving funds with external contracts.

Unlike State Channels, Arbitrum allows non-unanimous progress and doesn’t force large amounts of contract code to be executed on-chain in the case of disputes.

Unlike TrueBit, Arbitrum VMs are stateful smart contracts as opposed to stateless oracles. Furthermore, Arbitrum execution costs are constant rather than metered and independent of the amount of storage and code. Finally, Arbitrum does not require contract code to be public.

I’d welcome any questions or discussion here.