Do Rollups Inherit Security?

Thank you to Mike Neuder, Luca Donno, James Prestwich, Sreeram Kannan, Francesco D'Amato, and Davide Crapis for valuable feedback and review.

Introduction

Love it or hate it, Twitter will probably never stop fighting over "L2s" or whether rollups "inherit security".

While most of these debates are indiscernible semantic wars, the underlying points are incredibly valuable if you can manage to narrow them down. They get to the heart of when, where, and why rollups make sense. Do scalable L2s obviate the need for L1s? Is it possible to just turn L1s like Solana into L2s?

These debates largely come down to security

. Unfortunately, defining "security" here has been quite elusive. We mostly use the term casually where most people know roughly what we're talking about, but not quite. We'll break down security in detail here across different architectures.

Trigger Warning: Buzz Word Definitions

I'm sorry, but we've gotta get this out of the way quickly.

I finally get what ppl felt like when they didn't like my definitions of L1s and L2s and stuff

Just found out all of these are fruits:

Pistachios, cashews, almonds, olives, eggplants, peppers, zucchini, corn, green beans, peas, pumpkins, cucumbers, chickpeas

Absolute bullshit

— Jon Charbonneau (@jon_charb) July 2, 2023

Rollups

I've previously used the below definition from Mustafa:

"Rollups are blockchains that post their blocks to another blockchain and inherit the consensus and data availability (DA) of that blockchain."

And here's a more general definition from James Prestwich:

"A rollup is an opt-in, subset of another consensus, keeping a superset of state, via a custom state-transition function."

Neither require a validating bridge. The ability to build a bridge with minimal trust assumptions is a major benefit of rollups, but it's critical to analyze them in isolation.

We can consider the following criteria for rollups:

- Rollups are stateful systems (e.g., blockchains) derived by running a custom state-transition function (STF) over a data input on a host chain (the DA layer).
- All input data (i.e., full tx data or state diffs) used to derive the final confirmed state of the remote chain (i.e., the rollup) is confirmed on the host chain.
- Because the rollup state is derived from a STF run over data on the host chain, the rollup's validity is contingent on the host's validity. Rollup nodes must then fully verify the consensus and validity of the host chain(or make an honest majority assumption of the host chain).

Rollup nodes determine the rollup's state by applying the rollup's own STF over the outcome of the host's consensus (e.g.,

what data blobs the host confirmed the ordering and availability of).

Bridges

Bridges are systems which allow two blockchains to communicate with each other. Chain A (destination chain) needs to be convinced of some event on Chain B (source chain), and vice versa. Ideally, we want this communication to be bi-directional with strong associated security properties (e.g., high confidence that the message is valid, the source chain will not revert it, etc.).

Fundamentally, a bridge acts as an "observer" of another blockchain (just like any other typical human user). The bridge implements a given confirmation rule by which it is convinced of the state of the connected chain (e.g., how many Ethereum blocks must pass to accept an inbound transfer?).

- Traditional bridges often run onchain consensus-verifier light nodes of the source chain (i.e., they trust whatever the majority of consensus signs off on).
- A bridge can offer stronger security properties by acting as a full-verifier light node (i.e., adding data availability sampling (DAS) + validity/fault proofs). For example, the chain's validators may be required to all run DAS light nodes of a connected chain. This is a lighter alternative to requiring validators to run full nodes of the connected chain.
- Rollup bridges can also retain the liveness and re-org resistance properties of the host chain (because the rollup necessarily shares the host chain's consensus).

Bridging from host chain → rollup

This direction is pretty straightforward because rollup nodes fully verify the host. Rollup nodes know everything that happens on the host, so they know when a bridge transaction occurs. Current Ethereum rollup full nodes must also run full nodes for the Ethereum base layer itself.

Note that it would also be possible for rollup nodes to instead run full-verifier light nodes of their host chain if supported. Let's consider a hypothetical example where Ethereum has fully implemented the following upgrades:

- Ethereum execution blocks come with a validity proof. (There's research underway for a zkEVM at the base layer).
- Ethereum has implemented full DAS, so nodes can sample for DA.
- The Ethereum execution layer posts its data as blobs to the data layer, just like any other rollup on top of Ethereum. (E.g., Celestia's execution layer data will be posted to its DA layer, so DAS nodes will check the availability of rollup data and Celestia's own execution layer).
- Ethereum provides a full proof of consensus rather than relying on the sync committee (e.g., with a consolidation of validators, better signature aggregation, possibly ZK proof of consensus, etc.).

Now, say you want to run a full node for an Ethereum-based rollup. To follow the valid rollup chain, you must know Ethereum's canonical chain. This requires checking Ethereum's own consensus and validity:

- Ethereum's consensus
- Any light client can track what consensus has signed off on as the tip of the chain.
- Ethereum's own execution layer DA
- Rollup nodes will sample Ethereum's DA layer, checking the availability of rollup data and Ethereum's own execution layer data. (Note that DAS nodes do still make some additional assumptions over full nodes, which we'll see later).
- Ethereum's own state validity
- With a zkEVM in place, each Ethereum block would come with a validity proof.

The rollup node must check the state validity and DA of Ethereum's own execution layer because these are validity conditions for Ethereum blocks. The rollup node needs to know that it's not only tracking the Ethereum chain that consensus

has signed off on, but also that it's the valid

tip of the chain. For example, they could otherwise be accidentally tracking an Ethereum block that consensus signed off on, but is invalid (e.g., it prints a bunch of ETH).

If the base execution layer itself posts its data to the DA layer (just like other rollups) and adds validity or fault proofs, it would then be an enshrined rollup.

Bridging from rollup → host chain

This direction is trickier because the host chain is by default unaware of the rollup's state and STF (i.e., Ethereum nodes aren't required to run rollup nodes). To convince the host chain of the rollup's state, you can implement the rollup's logic in a smart contract (i.e., the rollup's validating bridge contract) deployed on the host chain. This contract checks the DA and state validity of the rollup.

Again, this bridge is optional. The smart contract on the host chain serves to convince all host chain nodes as to the validity of the rollup. This allows for two-way communication with favorable trust assumptions.

Rollups, Co-processors, & Intents

Based on what we just saw, is this a ZK rollup then?

If Cowswap added ZK proofs that solvers must submit with each batch showing optimal execution (by whatever pre-defined rules), is Cowswap a ZK rollup?

Solvers do complex offchain computation over many txs, batch them, prove results and settle back to L1, all data is on L1

— Jon Charbonneau (@jon_charb) September 4, 2023

As discussed, rollup nodes hold some state of their own (the rollup's state) in addition to the state of their host chain (e.g., Ethereum's state). So, does CoW Swap have its own state here that isn't part of the Ethereum state? If yes, then it sounds like a rollup. If not, then it may be a co-processor.

However, even this question isn't quite as simple as it might appear:

Arguably its not stateless in that the definition of optimal here is very much state dependent; that's in fact the crux of the problem OPT(bundle | state) vs. OPT(bundle) are of very different computational complexities

— Tarun Chitra (@tarunchitra) September 4, 2023

You may instead argue that the differentiating factor is the persistence of state:

Yes, but if the trades, prior to entering the solver, agree to be locked + transferred up to some limit, then you've effectively make a "just-in-time" L2 (assets are 'bridged' to the solver)

But yeah, I agree, let's not go into the semantic hinderlands; I just wanted to point…

— Tarun Chitra (@tarunchitra) September 4, 2023

What if CoW Swap permissioned certain actors to give users fast pre-confirmations (faster than Ethereum's block times), committing to the orders that will inevitably go into the batch, because the Ethereum batch times are longer than most users want? Now is it a rollup?

Chris Goes poked around this topic in his talk at Modular Summit. First, he offered an approximate definition for an intent as "a commitment to a preference function over the state space of a given system."

Note the parallels here between partial solving (matching intents) and rollup sequencing. An operator takes users' offchain signed messages → posts resulting data to the host chain.

- Intent-based application

- The resulting state change is settled onchain (e.g., in the CoW Swap example, the application lives on the base chain, and tokens are exchanged there as a result).
- Rollup application
- The resulting state change is computed for the rollup using the data committed to the host chain.

Intent-centric architectures and rollup-centric architectures approach similar goals from opposite directions. Intent-centricity broadly approaches this from the perspective of users and applications. Rollup-centricity broadly approaches this from the perspective of distinct blockchains.

Where you set the exact boundary on these isn't critical here. More importantly, we see that rollups are actually not so different from applications with offchain intent-matching that we're already used to

!

You rely on offchain actors (sequencers vs. solvers/fillers/etc.) for some weaker guarantees such as providing best execution and good UX → settle the results based on data posted to the host chain. However, they do not

custody your funds.

The line between the two may blur as verifiable offchain compute becomes increasingly relevant:

Basically same idea as trusting like UniswapX fillers not to screw you for example https://t.co/HEzEeQHw17

— Jon Charbonneau (@jon_charb) September 18, 2023

And if you want your intent solvers or rollup sequencers to be less trusted...

4) Any decentralized “intents” based trading system will need consensus for censorship resistance.

At which point it’s… an onchain CLOB.

— Elijah (.) (@Possibly_Real) October 5, 2023

Modular vs. Monolithic

Monolithic blockchains (a.k.a. integrated blockchains) are typically defined as chains which vertically integrate all core functions (i.e., consensus, DA, and execution). They're fully responsible for their own security. Solana and the Cosmos Hub are classic examples.

DA layers (e.g., Ethereum and Celestia) are often referred to as "modular" blockchains because they outsource execution to rollups, but this isn't quite accurate. They're also independently responsible for their own consensus, DA, and execution. Even Celestia will have limited execution (e.g., transfers, staking, bridging). Similarly, Solana wouldn't magically become a "modular" blockchain if someone launched a rollup on top of it.

So, when you hear people refer to chains like Ethereum or Celestia as "modular" blockchains, just realize this is more of a practical distinction rather than a strict technical one. Both are generally optimizing their own architecture to support rollups. These rollups are expected to handle the majority of execution in their domain.

Even rollups aren't necessarily entirely "modular." Rollup sequencers can reach consensus on transaction ordering, provide DA, and execute transactions prior to the host chain doing anything. This is how users get pre-confirmations. The host chain then provides another "final" commitment, again asserting DA and consensus over the rollup's transaction ordering.

Rollups vs. Integrated Chains

The more important distinction for our purposes here is "rollup" or "not a rollup." Is a chain's final state derived from data posted to a separate host chain (i.e., DA layer)?

While we associate DAS and validity/fault proofs with traditional rollups today, we should note that these are somewhat logically distinct concepts. Any “integrated” chain (e.g., a typical Cosmos app-chain) can in theory upgrade to add DAS and

validity proofs without posting its data to some other external host chain like Ethereum. Nodes would sample and check proofs for that chain alone.

Vitalik touched on this distinction in his classicEndgame

:

You may notice that a "traditional big block chain" (integrated chain) that adds DAS + validity/fault proofs can eventually just look like an enshrined rollup! Similarly, the "one rollup that scales and dominates" could become so successful that it just merges with its host chain to enshrine that rollup. The boundaries get fuzzy at the limit.

So, if you believe DAS + validity/fault proofs are the endgame, then "rollups" in some sense are inevitable. The distinction is effectively between the two approaches in the chart above:

- "Rollups" a.k.a. "Modular"
- Build logically separate chains which post data to their host chain (DA layer). They reuse the host chain's consensus.
- "Integrated" a.k.a. "Monolithic"
- Integrate everything into one protocol with its own consensus. Don't post data to a separate host chain. (Even if the DA layer and execution layer are in some sense logically separate pieces of the shared protocol).

When we discuss "rollups" throughout this report, we'll be referring to the former (i.e., not an integrated chain with DAS + validity/fault proofs, which may be referred to as an enshrined rollup).

While "traditional" rollups don't have a monopoly on DAS or proofs (i.e., integrated big block chains could add them), note that we're brushing over a lot

of technical details here. You can't just take Solana and decide "oh I think we'll add DAS today." This requires fundamentally rearchitecting the protocol in a way that starts to approach what we see Ethereum and Celestia doing:

The isse with tinydancer DASing is that Solana uses a different erasure code for each batch of 32 1280 byte shreds, so it's not actually DASing rn, as you'd have to do a DASing process for each batch

Requires fundamental Solana data structure change to work

— Mustafa Al-Bassam (@musalbas) June 2, 2023

Changing the way that data is encoded to support DAS would equate to slower block encoding and propagation speed, starting to look closer to traditional DA layers again:

The Firedancer figure on this chart is just the computational speed of computing the erasure code and Merkle root of the block. If you want to do an apple-to-apples comparison, Celestia should be 625MB/s on that chart (just benchmarked it on my laptop).

Firedancer is ~2x faster…

— Mustafa Al-Bassam (@musalbas) September 23, 2023

For this reason, we're seeing teams build the following:

- Specialized DA layers

(e.g., Ethereum's Danksharding, Celestia, etc.) - Slow blocks + DAS.

- Shared sequencers

(e.g., Espresso, Astria, or even Solana) - Effectively just fast DA layers without DAS.

However, fast blocks and DAS aren't necessarily incompatible

if you separate the timing of them. For example, you could imagine a chain like Solana offering two different paths:

- Fast path
- Continue to execute transactions and propagate data as fast as possible (just like today).
- Slow path
- Encode the data after the fact in such a way that it can be sampled asynchronously, providing DAS nodes with assurances slightly behind consensus.

This is something that Anatoly actually discussed in this podcast earlier today on how Eclipse can bring together Ethereum, Celestia, and Solana.

Coming from the other end of the spectrum, you could imagine a DA layer adding a faster path prior to making the data available for sampling:

mainly that we just need to get the first version out first

fewer blocks means LCs have to sample less

but if we decouple blocks from squares, then we just create a square once every X blocks

trust minimized finality is still just as long. honest majority is very short ~2s

— zkFART (@evansforbes) September 7, 2023

Providing two paths here within the same base layer protocol effectively internalizes fast shared sequencing, providing an interesting design for based rollups. Note that this is still an incredibly exploratory idea at the moment.

Confirmation Rules

Background in hand, we can now start to break down the security properties for these different architectures.

First off, nodes interact with any blockchain by running aconfirmation rule:

"A confirmation rule is an algorithm run by nodes, outputting whether a certain block is confirmed. When that is the case, the block is guaranteed to never be reorged, under certain assumptions, primarily about network synchrony and about the percentage of honest stake."

Arbitrarily many confirmation rules can exist for a given chain:

- How many blocks do you wait before confirming a Bitcoin transaction? 1? 6? 10?
- Do you confirm blocks based on Ethereum's available ledger usingLMD GHOST, or do you wait for the finality gadget (Casper FFG) to finalize?
- Do you run a full node that directly verifies every block, or do you only run a light node that checks what consensus signed off on?
- Do you just ask Infura?

Because each confirmation rule can make very different assumptions, they can have very different security properties even while interacting with the same chain:

(i) Security (both safety and liveness) is always a property of a particular confirmation rule, it is not a property of a chain.

— Sreeram Kannan (@sreeramkannan) March 6, 2023

This distinction is subtle but critical:

The insistence on focusing on confirmation rules is one of the few instances of consensus people being pedantic for a good

reason

— Francesco (@fradamt) September 3, 2023

Security = Safety + Liveness

Now let's dive into whether rollups then "inherit security" from their host chain.

Inherit

- It's perhaps clearer to say that rollups "rent" rather than "inherit" anything from their host chain. Rollups pay an ongoing cost for resources consumed (DA). Either party may choose to end this relationship. This isn't the interesting half of the question though.

Security

- We'll focus on security from here on out. The security of an algorithm is comprised of safety and liveness:
- Safety

(~nothing bad ever happens) - No two states ever deemed final by two properly-operating nodes will ever conflict.

- Liveness

(~something good eventually happens) - New states reflecting transactions suitable for inclusion will be finalized by all properly-operating nodes within bounded time.

Using Sreeram's great framing, we can break these down further into five properties that together make a confirmation rule secure:

Let's consider an example of a hypothetical integrated chain with DAS + validity/fault proofs. Its data is not posted to any other external host chain. For simplicity, we'll assume instant finality (e.g., Tendermint), so there's no available distinction here between an available ledger vs. a finalized ledger (e.g., as in Ethereum's Gasper).

We'll consider three confirmation rules that could be used to follow the chain using different types of nodes:

- Consensus-verifier light node
- Verify a proof-of-consensus (i.e., trust honest majority of consensus).
- Full-verifier light node
- Verify consensus + check DA (using DAS) + verify state validity (using validity/fault proofs).
- Full node
- Verify consensus + directly verify DA (download all data) and validity (execute all transactions and compute the state).

Confirmation Rules Have Security, Chains Don't

To emphasize this again, "[We colloquially talk about a chain

being secure, but really it is the confirmation rule

to which the security property attaches](https://twitter.com/sreeramkannan/status/1683735050897207296?s=20)."

Let's look at some examples.

CAP Theorem

For background, the CAP Theorem tells us that no single ledger can fulfill both conditions:

- Adaptivity (a.k.a., dynamic availability)
- remain live under dynamic participation (i.e., if a substantial fraction of nodes go offline), and
- Finality (a.k.a., consistency)
- remain safe under network partitions.

Consensus protocols tend to be grouped into one of two buckets, with each fulfilling one of the above conditions:

- Longest-chain protocols
- These protocols (e.g., Bitcoin's Nakamoto Consensus) guarantee liveness even when the number of actively participating nodes is variable (i.e., they're adaptive). However, they're not safe under network partitions (i.e., no finality).
- BFT-type protocols
- Classic consensus protocols (e.g., PBFT) achieve finality but not adaptivity.

Bitcoin Confirmation Rules

Bitcoin's consensus does not offer any hard economic finality.

Nodes observe the heaviest chain in their local view, and each user is free to apply whatever confirmation rule (e.g., accept blocks with >k

confirmations) they're comfortable with. The standard is to wait for 6 block confirmations, but it's up to you. It's reasonable to wait longer for higher value transactions.

There's a tradeoff between wait time vs. safety (i.e., likelihood of a re-org).

Ethereum Confirmation Rules

Ethereum's PoS consensus (Gasper) may seem to evade the CAP Theorem at first glance. However, it achieves both properties because it explicitly contains two nested ledgers

:

- Dynamically available ledger
- Safe and live under dynamic participation if the network is not partitioned.
- Finalized prefix ledger
- Always accountably safe. Remains live if the network is not partitioned and enough nodes are participating.

Gasper belongs to the family of "ebb-and-flow" (a.k.a. dual-ledger or dual-confirmation rule) protocols. Dual-ledger designs don't fall under the purview of the CAP theorem (i.e., it assumes a single confirmation rule). The finalized ledger falls behind the adaptive ledger when the network partitions, but it catches up when the network heals.

This allows the trade-off between adaptivity vs. finality

to be resolved at a user level

, rather than at a system-wide level

. This is a feature of "checkpointed longest chain" protocols as proposed in [Blockchain CAP Theorem Allows User-Dependent Adaptivity and Finality

](https://arxiv.org/abs/2010.13711). These protocols offer individual users the choice between finality and adaptivity instead of imposing it at a system-wide level.

Gasper clearly exposes two distinct confirmation rules, mapping to the two ledgers mentioned above:

- Dynamically available rule
- Guarantees adaptivity. Respects the tip of the heaviest chain. LMD GHOST is the fork choice rule used to determine the heaviest subtree.
- Finalizing rule
- Guarantees finality. Respects blocks confirmed by the finality gadget. Casper FFG is the finality gadget applied on top of the fork choice rule.

As discussed in the paper:

"The more optimistic adaptive rule always confirms blocks that are marked as finalized by the more conservative rule, and may possibly confirm more blocks during variable participation levels. Clients (users) make a local choice between the confirmation rules as per their personal preference, while miners follow a fixed block proposal rule that is consistent with both confirmation rules."

This allows for a system where all (honest) nodes:

- follow a common block proposing mechanism, but
- different nodes can choose between different confirmation rules.

Validators continue to extend the longest chain (mine new blocks at increasing heights) regardless of participation levels, but new checkpoints appear only if there is sufficient participation. The longest chain (containing the latest checkpoint) may alternate between divergent chains (i.e., re-org unfinalized blocks), but checkpoints are guaranteed to be on a single chain irrespective of network conditions (i.e., finality).

A user's security depends on what confirmation rule they abide by. There's a tradeoff between fast block confirmations vs. stronger safety guarantees. Users selling coffee may prefer liveness over safety, but users selling yachts may prefer safety over liveness.

Ethereum nodes could also apply some other intermediate confirmation rule heuristic for practical uses. Rather than using a naive k

-deep rule as in Bitcoin for the adaptive confirmation rule, we can instead add other heuristics which include assumptions regarding network synchrony and validator honesty.

That's exactly what is proposed in A Confirmation Rule for the Ethereum Consensus Protocol

. It proposes a confirmation rule with the following properties:

- In ideal conditions
- The rule will confirm a new block immediately after its slot.
- In typical mainnet conditions
- The rule should be able to confirm most new blocks in under a minute.

This confirmation rule is not a substitute for economic finality. Rather, it offers a useful heuristic to users who believe that network synchrony will hold for the near future. Let's compare the two:

Let's consider some examples. You're selling a yacht for $2.5mm ETH, and these are some of the possible confirmation rules:

- Full node + wait for finality
- Even a malicious majority of validators can't fool you into accepting an invalid block (e.g., printing fake ETH). If they pay you the $2.5mm then try to re-org that finalized block later, they'd suffer a huge cost (at least ⅓ of stake is

slashable).

- Full node + wait one block
- A malicious majority of validators still can't fool you into accepting an invalid block. However, they could send you $2.5mm ETH in a valid block, ride off in the yacht, then the block immediately gets re-orged. This is possible given enough stake weight or poor network conditions. They aren't slashed.
- Light client
- A malicious sync committee can lie to you with no penalty, and the buyer rides off in the yacht. (Note that this sync committee as a subset of consensus is unique to Ethereum. Other PoS chains with more efficient light client support can check all consensus votes given a smaller validator count.)
- Metamask
- You're just trusting Infura. The guy you're selling the yacht to promises the Infura employees they can take it for weekend rides, so they lie to you. You think you've got your $2.5mm, and you hand over the keys.

ethereum is an infura's L2

— donnoh.eth (@donnoh_eth) May 27, 2023

Rollup Confirmation Rules

As with any chain, nodes use different confirmation rules to interact with the rollup. The rollup's strongest confirmation rule will finalize alongside the consensus of its host. Rollup sequencers can expose weaker confirmation rules for better UX (i.e., fast pre-confirmations for impatient users), but users can also wait for the full security of the host chain's confirmation rules.

The typical rollup transaction flow works roughly as follows:

1. Users submit transactions to the sequencer
2. Sequencer orders transactions and gives pre-confirmations
3. Deterministic STF is applied to ordered transactions to compute the new rollup state
4. Updated rollup state commitment and associated transaction data are eventually posted to the host chain

Once the transaction data has been posted to the host chain:

- Rollup full nodes
- Directly verify that the proposed state of the chain is correct.
- Rollup light nodes (including the validating bridge)
- Unable to verify directly.

Different observers of the same rollup use different confirmation rules, so they finalize their view of it at different times:

1. Assuming full transaction data is posted (not just state diffs).
2. As noted earlier, rollup nodes must also run host chain full nodes or full-verifier light nodes (or make an honest majority assumption with a consensus-verifier light node). The rollup light node can be run as additional software or implicitly run inside of the host chain node (i.e., the bridge contract on the host chain verifies the rollup).

Users could also confirm transactions faster by trusting sequencer pre-confirmations even prior to the host chain receiving the data. Safety can fail if the sequencer misbehaves. Then once the data is on the host chain (and you've checked DA + validity), only a host chain fault (e.g., an Ethereum re-org) would affect your security.

So, even centralized sequencers don't really make "the rollup" less secure. You always get security up to your desired

confirmation rule. You can use the same confirmation rule (e.g., wait for host chain finality and check rollup validity) whether a rollup has a based sequencer or some other design. Assuming proper implementation (e.g., forced transaction inclusion via host chain), you can get the same security properties on the same time horizon holding all else equal.

Similarly, you could imagine Ethereum L1 block producers providing pre-confirmations because of the slow block times. That wouldn't make "Ethereum" less secure either. You'd just have another confirmation rule you decide whether or not to use (with lower security) until the higher security becomes available upon Ethereum validators finalizing it.

The idea of pre-confirmations fits very nicely with the logic underlying Gasper as described by Vitalik:

The general principle is that you want to give users "as much consensus as possible": if there's >⅔ then we get regular consensus, but if there's <⅔ then there's no excuse to just stall and offer nothing, when clearly it's still possible for the chain to keep growing albeit at a temporarily lower level of security for the new blocks. If an individual application is unhappy with that lower level of security, it's free to ignore those blocks until they get finalized.

Bringing this all together, we have a "zone of concordance" when all confirmation rules agree on the same state of the ledger at the same time:

When there are multiple confirmation rules, one can ask when the rules *eventually agree* - this defines a “zone of concordance.” The different rules can be thought of pulling the chain in different directions. If you operate outside the zone of concordance, the chain “explodes”.

— Sreeram Kannan (@sreeramkannan) July 25, 2023

Confirmation Rules - Security & Accessibility

You'll probably have worse security if your confirmation rule is to trust a single sequencer run by SBF vs. a decentralized sequencer set with the world's most reputable validators. Liveness failures and re-orgs are security failures.

Alternatively, you can just wait for the stronger (host chain) confirmation rule to become available. Then the untrustworthy sequencer doesn't impact your security holding all else equal. If you're selling coffee, you're probably good to go right away. If you're selling a yacht, you might wanna double check the host chain confirmation.

However, we can't totally ignore the potentially lower security of the "trust the random dude running a solo sequencer" confirmation rule if everyone actually uses that confirmation rule to sell their yachts. The precise design is a balance based on the level of progressive commitment needed at what time for the given use cases.

Similarly, this touches on what is more often the real criticism of high throughput blockchains like Solana. What confirmation rule do people practically have access to? You might have great security properties running a Solana full node, but that confirmation rule may be inaccessible

to most people (i.e., depending on resource requirements and/or cost).

Direct verification (i.e., not just trusting an honest majority) is a core property of these systems. So, we really care about two aspects here with respect to a given confirmation rule - security & accessibility:

DAS + proof style constructions have a heavy emphasis on making confirmation rules with near optimal security highly accessible

— Jon Charbonneau (@jon_charb) September 2, 2023

In summary:

- Users interact with any chain via confirmation rules.
- A single chain can have arbitrarily many confirmation rules.
- Security is a property of a confirmation rule, not the chain itself.
- We care about the security and accessibility of the confirmation rules for a given chain.

Practically speaking, when we say a given chain is secure, we're trying to express the notion that its relevant confirmation rules are both secure and accessible

.

Security in Rollups vs. Integrated Chains

1) Integrated Chain With DAS + Validity Proofs

We see now that security regarding DA and state validity can be checked directly via cryptographic techniques (DAS + validity/fault proofs) without strong assumptions on the chain's operators. Any protocol can technically implement these. Full nodes can also check DA and state validity with no external assumptions.

Let's focus on those other properties now - liveness and re-org resistance. As we saw before, these may fail regardless of which confirmation rule you're running

. Looking again at an integrated chain with DAS + validity proofs + PoS single-slot finality:

Selecting a strong confirmation rule is particularly effective against a subset of safety failures. Even a malicious majority of validators couldn't trick full nodes or full-verifier light nodes into all believing:

- unavailable data is actually available, or
- an invalid state transition is valid.

Full nodes are no more or less convinced of a block's DA or validity whether Ethereum has 1 validator or a gazillion. They're guaranteed by checking. Full-verifier light nodes can check in an even lighter way (though note that DAS will make some other assumptions that we'll discuss later).

However, a malicious majority of validators could halt ledger growth, censor you, or re-org the chain (which failures occur depends on the confirmation rule). DAS + ZK can't save you here. Re-org resistance and liveness are always reliant to some extent on various underlying properties regarding the given chain (e.g., reliable operators, economic incentives, social consensus, etc.).

It's less apparent here that liveness and re-org resistance are still properties of a given confirmation rule

, because every node is subject to the same attacks in the table above. They have the same assurances regardless of the confirmation rule here.

However, it becomes apparent again when you remove the single-slot finality assumption. In Ethereum's Gasper, you'd again have different liveness and re-org resistance properties depending on which ledger you follow (i.e., the available longest-chain ledger or the checkpoint finalizing ledger). A malicious majority of validators would cause different security failures depending on which confirmation rule you're running.

In any case, the point is the chain's underlying construction matters a lot here. You need robust operators, economic incentives, and social consensus to maintain the chain's liveness and re-org resistance. Additionally, dual-ledger consensus protocols such as Ethereum's provide valuable flexibility for users to make an availability vs. finality calculation for themselves based on their own needs.

2) Rollup With DAS + Validity Proofs

Let's slightly modify that example now:

- Previous example
- Integrated chain with DAS + validity proofs. Imagine taking Solana as it is today, but adding DAS + proofs.
- New example
- Rollup deployed on an external host chain (e.g., Ethereum) with validity proofs + DAS (note Ethereum DAS isn't live

yet). The rollup has a decentralized sequencer set which comes to consensus for fast pre-confirmations.

You'll notice that rollups then have two completely separate classes of confirmation rules for different time frames (i.e., whether you operate based on the sequencers' pre-consensus or wait for the host chain's final consensus). We'll look at each path now.

Fast Path - Prior to Host Chain Consensus

Rollup nodes can rely on the sequencers' confirmations (prior to posting to the host chain). We'll assume they can run the below rollup nodes:

- Rollup consensus-verifier light node
- Trust honest majority of the rollup sequencer consensus.
- Rollup full-verifier light node
- Run DAS on the sequencers' feed + check a validity proof before anything is posted to Ethereum.

(It's technically possible for rollup sequencers to facilitate DAS and provide validity proofs even before posting to the host chain, but in practice this doesn't happen. Full-verifier light nodes are generally intended to check these via the host chain. However, I'm assuming "real-time" DAS + proofs for a cleaner apples-to-apples comparison to the integrated chain.)

- Rollup full node
- Download all data + execute all transactions from the sequencers' feed to directly check DA and validity.

In the table below, these are the changes compared to the integrated chain example:

- You rely on the rollup sequencers

for liveness and re-org resistance instead of the integrated chain's validator set

.

- We simply removed theeventual liveness

properties, because we're only looking at the time frame prior to host chain consensus here. (These "eventual" liveness properties will later come from the host chain as we'll see shortly.)

Deletions are shown in red strikethrough, and additions are in blue:

Slow Path - Wait for Host Chain Consensus

For additional security, nodes can wait for the host chain's (e.g., Ethereum) consensus. This is where it comes into play more clearly that rollup nodes are expected to also run host chain nodes:

- Host chain consensus-verifier light node
- Trust the host chain's honest majority of consensus.
- Host chain full-verifier light node
- Check a validity proof for the host chain + run DAS on the host chain (which includes both the rollup data + the host chain's data).
- Host chain full node
- Download all host chain data (which includes checking the rollup data) + execute all host chain transactions to directly check validity.

Note that it's possible to then verify the rollup's state validity via two different paths:

- External to the host chain (run additional rollup node software)
- Rollups don't need their host chain to verify their state or STF. They don't need to deploy a validating bridge. Instead, you can just check the rollup's proof via another method (e.g., receiving the rollup proof via p2p). This requires running additional rollup node software to verify the proof (i.e., a rollup light node).
- Internal to the host chain (implement rollup node inside the host chain)
- This is the norm today. The rollup light node verifier logic is instead deployed on the host chain itself (i.e., the rollup's enshrined bridge contract). Because this rollup verifier node is run within the STF of the host chain, verifying the host chain's STF then implies verifying the rollup's STF.

If we get a ZK-provable host chain (e.g., Ethereum L1 zkEVM) + all rollups prove their state inside of the host chain → we get Vitalik's vision for a proof singularity. Verifying Ethereum's one ZK proof would imply verifying all others chains with verifier nodes implemented inside of it:

For simplicity, we'll assume here that the rollup's state validity is verified within the host chain itself (e.g., the rollup has an enshrined bridge with the host chain), so we can ignore explicitly running additional rollup node software out-of-protocol.

The changes from the previous "fast path" rollup table are noted below:

1. Achieved with forced transaction inclusion and/or forced sequencer/prover replacement via the host chain. We call this "eventual" liveness here as it's a slow-path from the perspective of the rollup, but this can be viewed as "real-time" liveness from the host chain's perspective.

Rollups vs. Integrated Chains

Now we can see the security property changes associated with integrated chains vs. rollups:

- DA & State Validity
- If implemented, DAS + validity proofs can provide applicable safety guarantees whether the chain is integrated or a traditional rollup. Practically speaking, these technologies are dominated by rollups today.
- Liveness & Re-org Resistance
- Integrated chains are independently responsible for these in all scenarios. Conversely, the rollup offers a choice of confirmation rules on different time horizons. You can take the less secure confirmation rule (trust the sequencer consensus) for fast guarantees, or wait for the more secure confirmation rule (wait for host chain consensus).

Liveness & Re-org Resistance

There’s no way to be cryptographically assured of these properties. One-shot signatures in a quantum setting may in theory be able to guarantee re-org resistance, but this is next level theoretical moon math for the foreseeable future).

Even across confirmation rules (e.g., whether running a full node or a light node), you may be susceptible to security failures here. No full node or ZK proof can protect you from a liveness failure or re-org if the chain operators go totally rogue.

These properties are achieved with robust and decentralized operators, censorship resistant mechanisms (e.g., inclusion lists), liveness-favoring consensus, high “cost” to re-org, strong social consensus, etc. These are often challenging to objectively compare - how do you measure operator decentralization or social consensus? There’s no one right answer. These are arguably the most difficult aspects to engineer, and they are indeed very unique to a given chain.

Importantly, we see that rollups can delegate re-org resistance and liveness to the host chain

. Confirmation rules used on the rollup can have up to the same security properties as if they were operating on the host chain at the same timeframe.

Chains can even pick and choose which properties to delegate to which chain. Different types of “L2” architectures (e.g., validiums, optimiums, and sidechains) can absorb different subsets of security properties. For example:

- Re-org Resistance
- A rollup may delegate re-org resistance to Ethereum. Its fork-choice rule is to select the "canonical chain" based on what Ethereum consensus confirms. If Ethereum re-orgs, the rollup re-orgs.
- Liveness
- However, rollup users still wouldn't receive Ethereum's liveness properties if the rollup lacks forced inclusion and forced operator replacement mechanisms.

It's also possible for a rollup to provide an escape hatch for users to exit out of the rollup, but retain the ability to censor users and prevent deposits into the rollup (e.g., Loopring works this way). If the deposit isn't processed after some time, the user can then withdraw their locked funds from the L1 contract.

This highlights the importance of such mechanisms:

On Arbitrum, the "escape hatch" (force inclusion) not only allows you to escape but also allows the DAO to elect a new sequencer

— Lee Bousfield (plasmapower.eth) (@PlasmaPower0) July 17, 2023

Data Availability & State Validity

Unlike liveness and re-org resistance, nodes can be assured of both DA and state validity without

making any large threshold assumptions (or without making a security tradeoff between the two). A malicious majority of block producers can cause liveness and re-org failures, but they can't cause DA or validity failures for full nodes or full-verifier light nodes.

However, consensus-verifier light nodes of course would be subject to state validity and DA failures from an honest majority. They just trust whatever consensus says. That's why DA and state validity are where making secure confirmation rules accessible

really shines. This is often the big difference in ideology between traditional rollups vs. big block chains that place less emphasis on user verification.

In order, these are generally the preferable ways to balance security and accessibility:

1. Make DAS and validity proofs broadly accessible.
2. If you don't have DAS and/or validity proofs, then make full nodes broadly accessible (i.e., low resource requirements, easy to run one, etc.).
3. If you don't have DAS and/or validity proofs, and full nodes are broadly inaccessible, then make consensus-verifier light nodes broadly accessible and have a trustworthy honest majority of consensus.
4. Ask Infura.

Note that #2 (full nodes) is actually the most secure. ZK verification is pretty straightforward, but DAS nodes make some additional assumptions that full nodes do not. However, they provide close to the same security as full nodes with a fraction of the resource requirements. They're scalable.

Full nodes simply download all the data, so they have 100% certainty. They only sign off on the block if it's all there. There's no assumption made on external parties.

The goal of DAS is to get security that's nearly as good as a full node with massively lower resource requirements (i.e., higher scale). This post on Security levels for data availability for light nodes covers this incredibly well.

In short, you're generally making some assumptions around network synchronicity and the existence of enough nodes to reconstruct data. Even a small honest minority of light nodes should be able to collectively reconstruct a block if an adversarial block producer withheld any data. There are also assumptions around selective share disclosure where an

adversarial block producer could trick some small amount of light nodes individually, but not collectively.

These “few of N” assumptions (e.g., honest minority of nodes to make DAS secure) are highly favorable relative to typical roughly “N/2 of N” assumptions (e.g., 51% of block producers can cause re-orgs), something covered very well in Vitalik’s post on Trust Models.

Overall, DA and state validity aren’t “delegated” by rollups in quite the same way as liveness and re-org resistance. DA and state validity are directly verifiable by the user, whereas the other properties more heavily rely upon the chain’s consensus participants and their incentives.

Looking back to the earlier example of where to verify the rollup’s proof:

- Post a rollup's ZK proof to an Ethereum smart contract. You run an Ethereum full node, implicitly verifying that proof.
- Send a rollup's ZK proof to my rollup light node. You directly verify the proof.

You’re assured of validity in either case. You’re no more certain of the validity regardless of where you check it. Ethereum isn’t really “enforcing” validity in the way that Ethereum nodes “enforce” re-org resistance or liveness properties. Re-org resistance and liveness are very dependent upon who you get them from.

Consider a rollup based on Scam Chain:

- The rollup's fork-choice rule is to follow the tip of the chain confirmed by Scam Chain. → If Scam Chain re-orgs, your rollup will re-org.
- The rollup's forced inclusion mechanism and sequencer removal are enforced via its bridge contract on Scam Chain. → If Scam Chain's ledger halts, then your rollup's ledger halts. If Scam Chain wants to censor your rollup, then you're gonna be censored.

Rollups have the ability to expose confirmation rules with equivalent security properties as their host chain. They can receive these properties at best at the speed of their host chain consensus

(and in practice it's often a bit slower depending on how frequently the rollup posts to the host chain).

Rollups can also make available a "happy path" looser confirmation rule (i.e., sequencers) for better UX, but they retain the fallback in event of failure. If your sequencer halts, you can keep moving. However, this is not the case if your chain is entirely reliant on your own validator set (i.e., as an integrated chain).

Choosing a rollup's host chain wisely makes a concrete difference in security properties. It's particularly valuable to leverage a host chain with strong liveness (ledger growth + CR) and re-org resistance properties.

Different Security Assumptions at Different Timeframes

Let's look at a simple example. Chain X (CX

) is deciding whether to deploy as a rollup on an existing host chain (CH

) vs. deploy as its own integrated blockchain.

CH

has the following features:

- CH

has 10s block times

- CH

can support DAS light nodes

- CH

can provide "high-security" confirmation rules with respect to liveness and re-org resistance (e.g., decentralized trustworthy validators, etc.)

CX

has the following features:

- CX

has 1s block times

- CX

can implement DAS light nodes and validity proofs whether it launches as an integrated chain or as a rollup on CH

- If CX

implements a centralized sequencer - it will provide "low-security" confirmation rules with respect to liveness and re-org resistance

- If CX

implements its own decentralized consensus (either as a rollup sequencer set for pre-confirmations or as an integrated chain validator set) - it will provide "medium-security" confirmation rules with respect to liveness and re-org resistance

The table below gives a simplified visual representation of the best security guarantees that CX

users could have under various implementations (i.e., they're using the strongest confirmation rule available). In particular, we're looking at liveness and re-org resistance here (since we gave the benefit of the doubt assuming CX

will just implement DAS + validity proofs in both scenarios):

The rollup's "eventual security" is higher than its "real-time security" here because the host chain can't give us guarantees faster than its own block time. Even if you can check that the sequencer's pre-confirmations are valid state transitions, you don't have full assurances of their finality until they end up on the DA layer.

As we can see though, deploying as a rollup to a robust host chain can be additive to security. They can rent security at the speed of their host chain

. There is fundamentally no way to get the full security properties of the host chain at a speed faster than the host chain's own consensus.

However, users are often impatient. So, rollups typically offer faster pre-confirmations with lower assurances in the interim. A rollup can choose a sequencer design that balances:

- Practical features & efficiency
- E.g., centralized sequencers can provide fast pre-confirmations and reduce operational overhead.
- Robust guarantees
- E.g., an incredibly decentralized sequencer set may provide better real-time liveness, at the expense of higher operational costs and latency. (At the extreme end, you just let your DA layer handle sequencing for your rollup, choosing not to expose a faster path).

Interestingly, you can argue that L1-sequenced rollups have worse

liveness than a centralized sequencer depending on your timescale

. We've discussed liveness so far as getting inclusion within some notion of "bounded time." However, that's entirely relative and subjective. Relative to what? What amount of time?

A naive L1-sequenced rollup will only ever be included at the speed of the L1 blocks (e.g., 10 seconds). You have no liveness guarantees between those blocks, all while the world is changing around you. So it depends on your benchmark:

- If benchmark = actions internal to your rollup
- An L1-sequenced rollup may provide better liveness. Others on your chain shouldn't get a commitment prior to the host chain confirming it, so you're all on even footing.
- If benchmark = actions external to your rollup
- A rollup with soft pre-confirmations can provide better liveness. Pre-confirmations are just a free option. You still fall back to host chain guarantees at the host chain's speed, but you can get a weaker guarantee in the interim. If you don't trust them, just wait for the host chain to confirm. The world doesn't freeze in between Ethereum blocks. Stale prices between long block times may be unacceptable for many applications.

If you tried to implement a "true" based rollup without pre-confirmations, it's even possible that pre-confirmations would arise regardless. There's a financial incentive for actors (e.g., Ethereum builders and validators) to give this promise themselves. That's exactly why there's discussion of how Ethereum builders and stakers could try to give fast pre-confirmations even at the base layer.

One last important note here. The assumption that rollup users can fall back to equivalent

liveness as the host chain assumes that you can get forced inclusion at exactly

the speed of the host chain blocks (e.g., if the rollup sequencer is censoring you, that you can force transaction inclusion in the next Ethereum block).

In practice, there is generally a short delay. If you allowed for immediate forced inclusion, you could expose profitable censorship MEV among other complexities. However, there are designs which may provide near real-time liveness guarantees from the host chain (e.g., perhaps at the speed of a few host chain blocks rather than one block).

Regardless of the exact time scale, absorbing the host chain's eventual liveness is incredibly powerful. Using a robust host chain as a coordination mechanism provides a credible threat and right to exit. Just exposing this credible threat in and of itself makes it exceedingly unlikely for it to ever be needed, preventing the malicious behavior in the first place.

For example, a centralized rollup sequencer can't just arbitrarily extract rent from its users and lock them in if they have a credible mechanism by which they can forcibly exit and even forcibly remove the operator. This is a general area which Chris Goes discussed in his presentation on The Edge of MEV Switching Costs and the Slow Game at Research Day.

Accidental liveness failures can also of course happen, in which case this backup path can again be quite valuable.

Proofs Don't Secure Chains, They Secure Users

Following all this, we can see that proofs more accurately secure different "observers" (users) of a rollup for a given confirmation rule rather than securing "the rollup" itself. The security of "the rollup" does not exist as a single concrete measure.

Keeping observers secure is of course what practically matters because we're all observers of the chain! The chain is whatever its observers say it is. If you can't observe it in a secure manner, you'd have to trust others (e.g., validators) to tell you the "truth" of it. But we don't want to trust, we want to verify → we want proofs.

To see why delineating between "proofs secure the chain" and "proofs secure observers of the chain" is important, consider the following:

- Light node
- A rollup light node is more secure if it has proofs. They don't have to take anyone's word for its validity.
- Full node

- A rollup full node is no more or less secure if it has proofs! You could launch a rollup without an enshrined bridge or even any proofs (a "pessimistic rollup"). A full node would be no more or less secure if you started sending around validity proofs:

I don't know who needs to hear this, but pessimistic rollups are real.

They're rollups that only support full nodes.

Basically how Tether uses Bitcoin as a DA layer.

They're not great for scaling as they don't scale validity checking, but can be used for app-chains.

— Mustafa Al-Bassam (@musalbas) February 24, 2023

Proofs add security to observers of the chain who are unable to directly check its validity (i.e., light nodes). We don't want users to have to run beefy full nodes, so these proofs are important.

If dishonest majority signs an invalid block:

- it will not trick any user having a full node
- it will not trick a rollup validating bridge
- it will trick user running light client
- it will trick a light client bridge or zkBridge (e.g. IBC)
- who knows what MSig bridge will do

— bartek.eth (@bkiepuszewski) September 2, 2023

Proofs Secure the Rollup's Bridge

The rollup's validating bridge is one incredibly important observer! Proofs do secure the bridge!

Like any typical light node, the bridge is unable to directly check the rollup's validity. Rather than trust an honest majority, we protect the bridge with proofs. The consensus protocol of Database A (the DA layer) orders data blobs, then the validating bridge independently checks the validity of the corresponding updates to Database B (the rollup):

A bridge is an observer of another chain, and every asset it mints always carries the security assumptions of the bridge's confirmation rule

. The security of its confirmation rule can have wide-ranging effects. That's why building secure bridges is so important! (Or ideally reducing the need for so many bridges in the first place by scaling single-chain execution further.)

A malicious party can't trick you into accepting an invalid state transition if you run a full node for that chain. However, a malicious party could still trick a bridge

if it has a different confirmation rule. If you hold assets backed by collateral in that bridge, then your funds could become unbacked. Security failures which trick bridges are "contagious" in this sense.

Let's consider an old hypothetical scenario with Terra:

- Terra has its own validator set, its native token is LUNA, and it can issue native UST.
- Osmosis has its own validator set, and its native token is OSMO.
- We have a standard Terra ←→ Osmosis IBC bridge where each side runs a consensus-verifier light node of the other chain (i.e., each side of the bridge relies on an honest majority of the other chain's validator set).
- You as a user run your own full node of each chain.

- We'll call UST on Osmosis bridged via IBC = osmoUST.
- We'll call OSMO on Terra bridged via IBC = terraOSMO.
- You own terraOSMO on Terra.
- You're LPing in the osmoUST/OSMO pool on Osmosis.

As Terra collapsed, the price of LUNA plummeted. Eventually, the theoretical profit from the validator set turning malicious would surpass the value of LUNA staked. Terra validators could've signed off on invalid blocks which did the following:

- Print fake UST → bridge to Osmosis to mint osmoUST. Use this to drain all osmoUST trading pairs (e.g., taking all of the OSMO from the osmoUST/OSMO pool).
- Print fake terraOSMO → bridge to Osmosis to withdraw all of the native OSMO collateral locked on Osmosis that backs terraOSMO. All terraOSMO left on Terra would now be unbacked.

So, what security failed here:

- Full node (me)
- My full node recognized the Terra blocks as invalid and rejected them. Terra validators couldn't just steal my terraOSMO or osmoUST/OSMO LP position.
- Light node (bridge)
- The bridge simply checked that Terra's consensus signed off on the blocks (without checking for a valid state transition), so it did not reject them. Terra validators could just steal the OSMO collateral backing my terraOSMO and drain all OSMO from the osmoUST/OSMO pool (leaving a bunch of worthless osmoUST).

The bridge was using a confirmation rule with a much stronger trust assumption.

The Terra validators would not have been able to steal a bunch of assets on Terra itself from full nodes (they wouldn't have been tricked). They would reject these blocks. However, the validators could've tricked consensus-verifier light clients (including the bridge). That's why consensus faults can impact assets across chains.

(Note that these faults are importantly not "contagious" to the rest of the chain. I.e., this fault is "contagious" to Osmosis assets exposed to the Terra bridge route, but there's no safety fault for the Osmosis chain itself.)

However, we obviously want to bridge stuff (and generically, communicate cross-chain). Being limited to only using native assets on their home chain would suck. We need chains to communicate and bridge securely.

As a thought experiment, the naive solution is just to have every user run a full node of every chain. This includes the bridges themselves. Remember, we already see some one-direction embedded full node bridges:

- Ethereum rollups all currently require their nodes to run Ethereum full nodes. These rollups share Ethereum's consensus.
- Namada (a separate Tendermint chain, not a rollup) will require its nodes to run Ethereum full nodes. However, Namada doesn't share Ethereum's consensus (i.e., it doesn't post data to Ethereum or derive its state based on that).

This works with an Ethereum full node, but this obviously doesn't scale. You couldn't start having every Cosmos chain just require running a full node of every other Cosmos chain. These bi-directional full node bridges wouldn't be scaling, they'd just be naively increasing hardware requirements.

Luckily, there's a more scalable option. We can deploy bridges which fully verify the state validity and DA of the other chain (in addition to still checking consensus).

State Validity

- While IBC is used today with traditional consensus proofs, it could be modified to add validity proofs for the connected

chain. Now the bridge wouldn’t be tricked by an invalid state transition. This would’ve prevented exactly the attacks described earlier. The bridge would have rejected the Terra validators’ block as invalid if it was able to verify the state transition validity.

This would look quite similar to how rollup bridges on Ethereum check a proof for the rollup, except these could also be bi-directional here.

DA

- Additionally, it’s possible for chains to require their nodes to run DAS light nodes of connected chains. For example, you could have these two Cosmos chains require their own validators to run a DAS light node of the other chain (if each chain implemented DAS light clients, which these chains don’t support today). With this in place, each chain would now know the validity and DA of each other, without running a full node.

For rollups, you by definition have all the data on the host chain, so the bridge there has access to it. In the other direction, rollup nodes run host chain nodes.

Dependent vs. Independent Chains

Let's look at our five security properties one more time, now in the context of a rollup's validating bridge on Ethereum observing the rollup:

- Ledger growth
- Ethereum validators can enforce that the rollup's ledger continues to grow (e.g., with forced transaction inclusion).
- Censorship resistance
- Ethereum validators can enforce that the rollup's operators will not censor indefinitely (e.g., with forced transaction inclusion or sequencer replacement).
- Re-org resistance
- The rollup's re-org resistance is tied to Ethereum. If Ethereum re-orgs, then all rollups on Ethereum will re-org together.
- Data availability
- The rollup's DA is ensured because the rollup is by definition derived from the data confirmed by the host chain's consensus (where the rollup contract is). The rollup and the host chain share a merged consensus. DA is a validity condition for Ethereum itself, so if data wasn't available then the Ethereum block would itself be invalid. The bridge can access data on the host chain with no added trust assumptions.
- State validity
- The contract will check a validity proof (or wait for the challenge window to pass) for the rollup's state transition. This proves that the claimed state updates are valid results of applying the rollup's STF over the corresponding data confirmed on the host chain.

Note that the bridge's security isn't just maximized by having a super strong confirmation rule of the attached chain (e.g., a full-verifier bridge to a chain with a super trustworthy validator set). You get the max security in bridging a native asset if the bridge carries the same security assumptions as the home chain.

Vitalik offered a helpful illustrative example:

"You move 100 ETH onto a bridge on Solana to get 100 Solana-WETH, and then Ethereum gets 51% attacked. The attacker deposited a bunch of their own ETH into Solana-WETH and then reverted that transaction on the Ethereum side as soon as the Solana side confirmed it. The Solana-WETH contract is now no longer fully backed, and perhaps your 100 Solana-WETH is now only worth 60 ETH. Even if there's a perfect ZK-SNARK-based bridge that fully validates consensus, it's still vulnerable to theft through 51% attacks like this.

For this reason, it's always safer to hold Ethereum-native assets on Ethereum or Solana-native assets on Solana than it is to hold Ethereum-native assets on Solana or Solana-native assets on Ethereum.

And in this context, "Ethereum" refers not just to the base chain, but also any proper L2 that is built on it. If Ethereum gets 51% attacked and reverts, Arbitrum and Optimism revert too, and so "cross-rollup" applications that hold state on Arbitrum and Optimism are guaranteed to remain consistent even if Ethereum gets 51% attacked. And if Ethereum does not get 51% attacked, there's no way to 51% attack Arbitrum and Optimism separately. Hence, holding assets issued on Optimism wrapped on Arbitrum is still perfectly safe."

You can replace Solana with whatever chain you want - even a hypothetical chain where you trust the validators even more than Ethereum's. Fundamentally, any security assumption is additive when you're talking about bridging an asset from its ledger of record (e.g., ETH from Ethereum), and there's always a chance the connected chain can re-org or have a liveness failure.

However, chains which have a merged consensus (i.e., rollups sharing their host chain consensus) can sidestep these additional security assumptions. Bridges between these domains can have the same eventual liveness and re-org resistance properties as the host chain itself. Shared consensus minimizes the bridging trust assumptions within this shared security domain.

What's a reasonable security assumption is for you to decide then. In practice, there have been no consensus failures among major chains like this. It would be plainly visible and costly, but this could be a stronger assumption for bridging to weaker chains.

There are also downsides associated with tying your rollup chain to a host chain. Let’s compare two bridging scenarios, where in both cases we have two chains using bi-directional full-verifier bridging:

- Dependent
- The remote chain (i.e., rollup) shares the consensus of the host chain (i.e., DA layer) and derives its own state from data on the host. The remote chain necessarily forks together with the host and bases its finality off of the host chain.
- Independent
- These chains have their own independent consensus, and they do not derive their own state based on data on the other chain. They do not share a remote chain (i.e., rollup) ← → host chain (i.e., DA layer) relationship. They do not re-org together or depend on the other’s liveness.

Interestingly, these are both good and bad:

- Bad
- Tying your chain to re-org with some other chain or have liveness failures may sound like a downside

at first glance. Why should my chain have problems just because your chain screwed up?

- Good
- This is actually an essential benefit

if such a divergence would cause massive problems on your chain (e.g., if you have a ton of bridged assets from the source chain, then it re-orgs away from your bridge's view of it, leaving unbacked collateral). Chains and their bridges need a consistent view of each other.

Similarly, there are potential positives and negatives regarding lock-in and excessive rent-seeking. This highlights why a neutral and censorship resistant base layer is so important:

- Good
- A good host chain can protect rollup users from the rollup operators extracting excessive value from them, enforcing the right to exit.

- Bad
- A bad host chain can just arbitrarily jack up prices and extract this value from the rollup and its users itself.

There's also some inefficiency in submitting proofs + running DAS bi-directionally rather than sharing a common host chain:

- You don't want to require your nodes to run DAS light nodes of a bunch of other chains and manually have to add new chains. Running DAS on one giant shared DA layer is far more efficient.
- It's inefficient for each chain to be verifying validity proofs bi-directionally for many different chains. However, it would in theory be possible to aggregate proofs among many chains such that only a single proof needs to be posted onchain for a whole cluster of chains.

There are tradeoffs and benefits here, but also keep in mind that there's a huge path dependency

of where the interesting assets are here. If you want to use a bunch of Ethereum-native assets (including those in its rollups), then rooting your chain around Ethereum and its bridge there makes sense:

it is not possible to build a secure bridge across two entirely separate blockchains that don't share security

it's not fanatic fealty to eth, it's a desire to use the stuff that's already on there, no other chain comes close to having those things

— c-node (@colludingnode) April 18, 2023

Conclusion

"Rollups inherit security" is nice shorthand, but remember these are the key points we really mean:

- Rollups pay rent to their host chain (e.g., Ethereum) for the resources they consume (DA).
- Rollups can expose confirmation rules with security properties up to those of the host chain (i.e., users can get up to the same security guarantees that they would have if they were to operate on the host chain itself).
- Users get these host chain security properties at the speed of the host chain's consensus.
- If rollup users want confirmations faster than the host chain can offer, they may use a confirmation rule which makes additional temporary security assumptions.
- Observers (i.e., users) interact with the rollup via confirmation rules. A validating bridge with minimal trust assumptions is one such (optional but incredibly valuable) observer.

Now, considering the benefits of deploying a rollup over an integrated chain:

- User Security
- Regardless of whether you want to implement any bridges, rollups can rent DA from their host chain and recycle its consensus. This allows rollups to expose confirmation rules to their users which match those of the host chain. No need to spin up your own consensus.
- Bridge Security
- Rollups can build bridges within the host chain's shared security domain (i.e., the host chain itself and its rollups) with security properties comparable to operating on the host chain itself.

We should see that these are actually two sides of the same coin. Rollups just allow a chain to provide confirmation rules with security up to that of the host chain. Bridges and normie users are both observers of the chain with a given confirmation rule. Bridges are likely the most important "observers" of these chains, as their security has wide-ranging implications.

When we're trying to make secure systems, we need to care about both the security of given confirmation rules as well as the accessibility of them. Secure confirmation rules aren't very helpful if they're out of reach for most observers (users)

interacting with these chains.

While we associate DAS and validity proofs with rollups today, they are logically separate concepts. Any chain can integrate these technologies, and the distinction of what is or isn't a rollup starts to break down at the Endgame (i.e., for a single integrated enshrined rollup that may have logically separate DA and execution layers under one protocol). However, "traditional rollups" and DA layers are clearly dominating these verification and scaling technologies today.