All the stuff in Chromia

Basics

We start with very basic stuff but I promise it gets more interesting later. We need common ground to establish more advanced concepts.

Software component: a piece of software, such as a **library**, **service** or an **app**.

Library: a software component which is meant to be used by many developers, e.g. integrated into a concrete application. A big library might be called a **framework**. (Typically the distinction is: you call a library; framework calls you.)

API (Application Programming Interface): Description of points through which a developer can interact with a library or a service. I.e. interface between a library and the application using it.

- Library API: description of functions, classes and other elements of a library which application developer is supposed to use. Usually it's the documented part. Elements which aren't part of the API might be considered internal, and might change more often.
- Service API is usually a description of "endpoints" and communication protocol, e.g. "to achieve X you can call endpoint Y with parameters Z encoded as W".

ABI (Application Binary Interface):

- In general, it's a description of calling conventions, i.e. a detailed description of how function calls are made, how parameters are encoded, etc. I.e. a particular compiler might implement a particular ABI. Application developers might never really deal with ABI beyond selecting a toolchain.
- In Ethereum, ABI is a "smart" contract's API. Why is it called ABI then? Because functionally it describes how to encode parameters. But, in practice, only Solidity calling convention is being used, and that calling convention is the ABI in the traditional sense. What Ethereum people call "ABI" is a smart contract's interface. In fact, if you plan to call a contract from Solidity, you only need an interface, not ABI. It's the same thing as the interface but in a different format.
- Does Chromia have an ABI? No, not particularly, but you might call GTX an ABI. (See below.)

Standard: something many people agree on.

Protocol: A standard for something which involves communication. E.g. it would define how to encode messages, what messages mean, etc.

- Can we call ABI a protocol? Yes. If it's an important API, which is of interest to many people, it might be, in a sense, a standard. Thus it can be called a protocol.
- In crypto, there's a tradition to call your smart contract's interface a protocol if it's meant to be used by many apps. Or just to make the thing you're doing sound bigger.
- **Implementation** of a protocol is a software component which implements the protocol, in some sense. Some protocols have a single implementation. Others might have many different implementations.

FT4 and friends, part 1

We call FT4 a protocol because it's meant to be used by many software components and applications. (Ideally, everything on Chromia should use FT4.) The implementation of FT4 is multiple things:

- FT4 Rell library is the implementation of the blockchain side, and is currently the only one.
    - FT3 Rell library was modified by some applications, e.g. we have FT3.5
- FT4 Client libraries used by the application front-end to interact with blockchains using FT4.

When you talk about FT4, make sure you know which part you're talking about: the protocol (i.e. the standard which describes how transactions should be encoded), the blockchain library or the client library.

GTX is a protocol for encoding Chromia transactions. FT4 transactions are encoded using GTX. We rarely talk about GTX because for most people it's just how you interact with Postchain/Rell. **postchain-client** library implements the GTX & Postchain REST API. Going even deeper, **Gtv** is the encoding format used by GTX. Gtv itself is based on ASN.1.

Postchain is a blockchain framework which implements a blockchain node.

Chromia Core is a combination of **Postchain** and **Rell**.

[Crypto] Wallets

The definition of a crypto currency wallet is not trivial. In fact, many people proposed to use a different term because it's so confusing, but we still ended up with "wallet".

Auth(???)

Let's start with dictionary definitions:

Authentication: The process of verifying the identity of a user.

Authorization: The process of granting a user access to specific resources or services.

[Authentication] credentials: pieces of information that are used to authenticate a user's identity.

Authentication is fundamentally not a useful concept in the context of blockchain programming. User is something which exists in meatspace. Neither blockchain nor client software has direct access to meatspace, it doesn't interact with the user directly, thus it cannot verify the user.

Authorization is a useful concept, though: A user authorizes transfer of money, and blockchain can verify the proof of authorization. It still doesn't have access to the user, of course, but it can define what sort of a proof is sufficient. E.g. it might be a digital signature made using a particular keypair.

So, if we talk about auth, in 99.9% cases we mean authorization.

In crypto, authorization is usually done by signing a transaction with the user's private key.

Crypto wallets

In the context of crypto, wallet has two meanings:

- A container of private key material (e.g. a hardware wallet). It's a thing which can be used for authorization.
- An application (UI) which facilitates creation and authorization of transactions, as well as related functions, such as
    - List of assets available to the user
    - Transaction history

Also, a lot of users believe that a wallet keeps their assets. They take an analogy from the real world: if you put cash into your wallet, that's where the cash is. If the wallet is stolen, cash is gone. (But consider a situation where you put a credit card and an identity card into a wallet.)

This is not entirely without merit. Wallets often contain the client part of the token transfer protocol, so tokens might not exist outside of the wallet. In some cases (e.g. LN) wallets keep some information crucial for asset retrieval, so assets are literally stored in the wallet's state.

But typically, with protocols such as Chromia's FT4 and Ethereum's ERC-20, tokens are stored in the blockchain, not in the wallet. A wallet can display tokens. But a block explorer can also do that - they query this information from the blockchain.

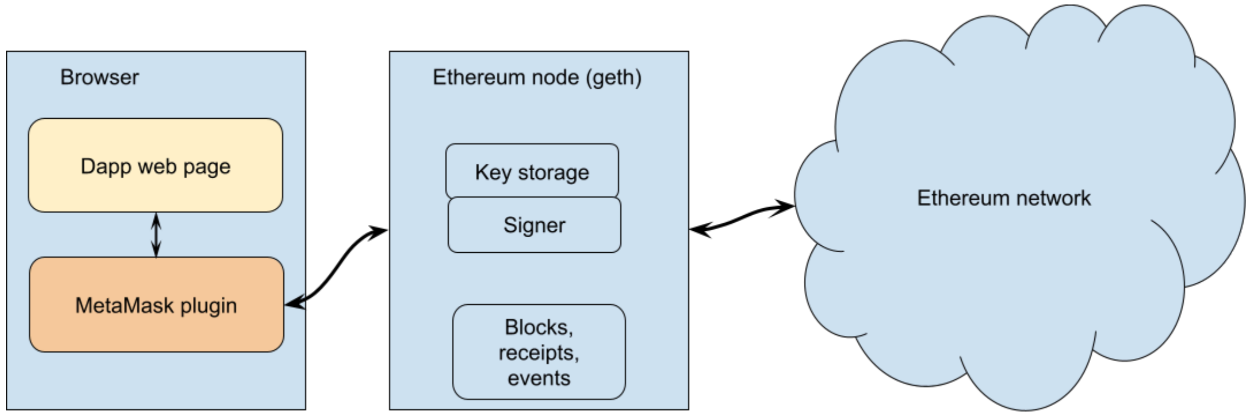
Generally, a wallet might take *some* of the following functions:

1. Key storage
2. Transaction & message signing
3. Transaction creation
    a. Transfers
    b. Bridging
    c. Swaps
    d. "Smart contract" / "dapp" interaction
4. Transaction history view
5. Asset view

MetaMask

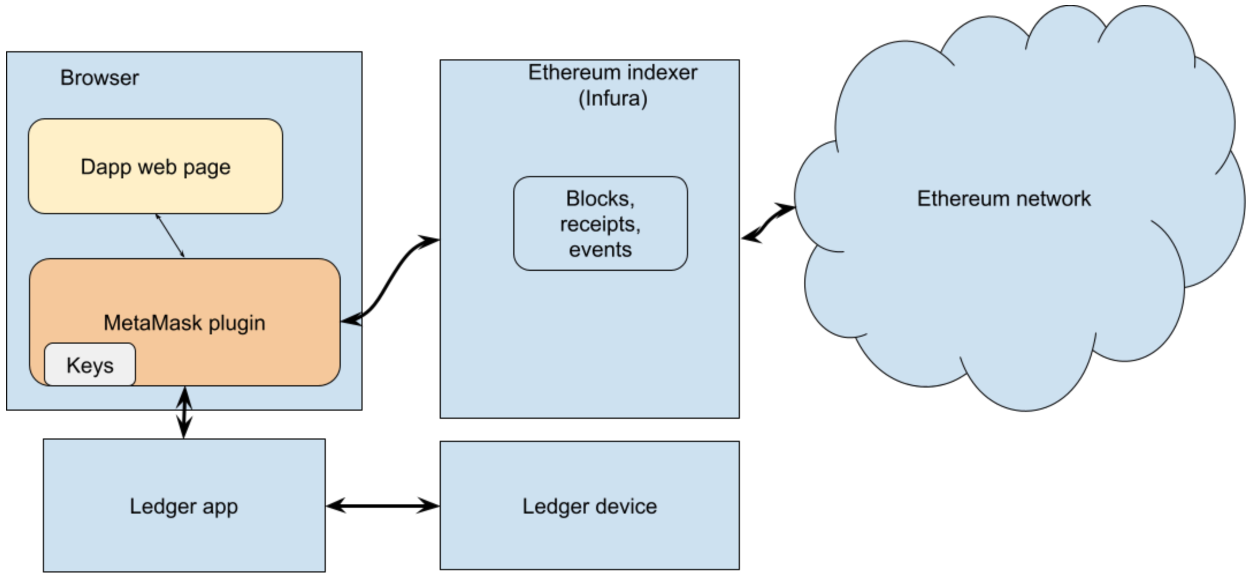
MetaMask was initially designed as a *connector* between a dapp front-end which creates a transaction and transaction signer. It also provides access to block & event queries.

Here's how it worked initially:



Then key storage & signer moved to MetaMask itself. This way one node can serve many users.

Later Infura offered access to their nodes to MetaMask for free, so now it typically looks like this:



Users can choose between storing keys in the MetaMask (i.e. in the file accessible by the browser) and using some other signing method, such as a Ledger hardware wallet.

The role of MetaMask is essentially to connect the dapp front-end to

1. The node which indexes information from the network (and can push transactions to other nodes)
2. Signing method, possibly one which is provided by MetaMask itself.

MetaMask ability to display a list of assets is NOT essential. It's not important.
A user might go to app.mycrypto.com to see their assets and send transfers. Or they can go to myetherwallet.com. They can transact using tokens which MetaMask doesn't know of.

In fact, for a long time MetaMask only supported ERC-20, but not ERC-721 and ERC-1155. And yet people could transact ERC-721 and ERC-1155 tokens using dapps which understand those.

"The wallet is not the wallet"

For a while I've heard something along the lines of: "When are you gonna release the wallet? I need it to make it possible for users to use tokens!" from Chromia dapp developers.

No, you don't need "the wallet" to add tokens to your dapp. The wallet is not the wallet.

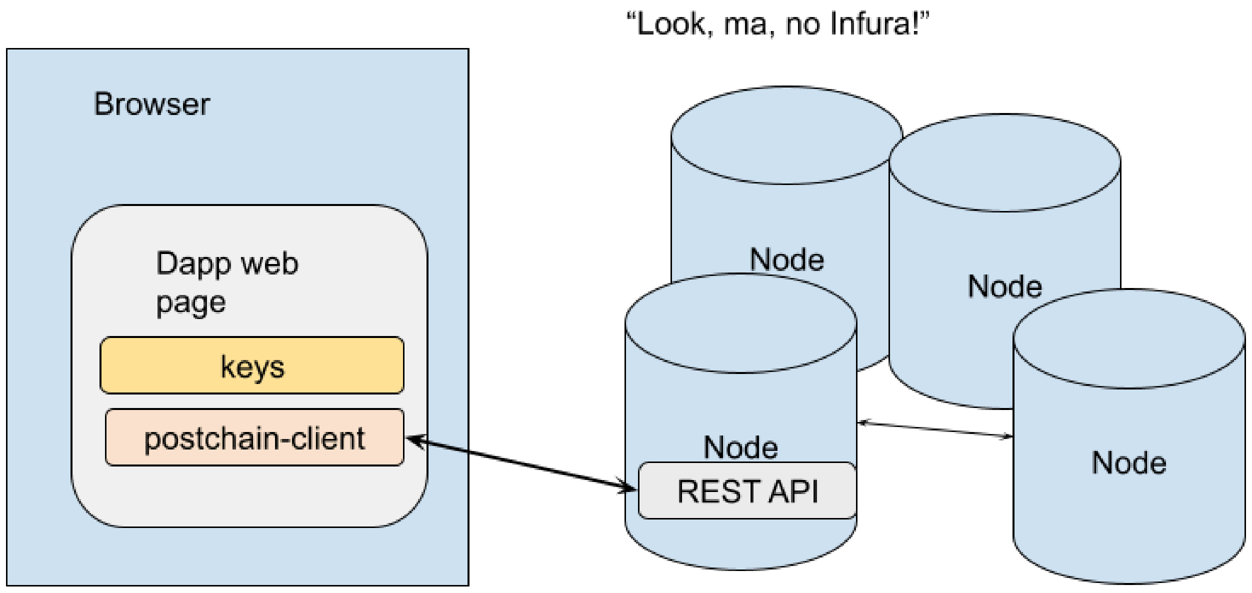
Strictly speaking, what you need to add tokens to your dapp is:

- Token implementation in Rell, e.g. FT4 or FT3 library in Rell

- **postchain-client**
- Understanding of FT4 API

Having a FT4 client library can also help. But notice how you don't need "the wallet".

The reason why something like a MetaMask plugin is not strictly necessary for Chromia is that a web page in the browser can directly connect to Chromia nodes via REST API. Ethereum nodes do not provide this, so apps have to connect to Infura using MetaMask. But this is not needed for Chromia.



But what's about key storage? Key storage is definitely useful.

But the thing is, most of our users probably already have MetaMask installed. So instead of forcing them to install one more plugin, we can just use MetaMask as something which connects us to key storage/signer.

FT4 + MetaMask

One of the new features in FT4 is "MetaMask support". (Or, rather, Ethereum signature support.) What we mean by that:

1. FT4 blockchain protocol can recognize Ethereum-style message signatures:
    a. We translate FT4 operations (i.e. parts of a transaction) to plain text, e.g. "Transfer 100 CHR to abcdef11222" is literally written as text.
    b. We request web3 API (MetaMask) to sign that text.

    c. When Chromia transaction is processed, FT4 code can understand that Ethereum-style signature is being used, then it translates operation to plain text and checks whether the signature is correct.
2. FT4 has Ethereum “authorization descriptor” which says which Ethereum address can sign transfers for a particular FT4 account.
3. FT4 client library can invoke web3 API message signing request, which goes to MetaMask, for example.

Note:
- User will only see the message to sign in MetaMask
- User will not see their Chromia chain balance in MetaMask
- We do not make Ethereum transactions. Message is in plain text.
- This is not connect to any Ethereum network

This works with any web3-compatible Ethereum wallet, WalletConnect, etc. Not just MetaMask. Should work with Ledger, etc.

Any dapp can use Ethereum auth directly. There are several modes:

- Ethereum auth is used to authorize every transaction. This is relevant when you deal with infrequent, high-value transactions.
- Temporary key is created by the dapp front-end to sign multiple transactions. (E.g. can be used for a game, DeFi, etc.)
- A dapp key is created by the dapp and stored in localStorage. This is more convenient but less secure.

Chromia Vault

OK, then why do we need Vault?

Strictly speaking, we don’t. Dapps can work without Vault. Previously we recommended dapps to use SSO provided by the Vault, but it’s cumbersome and rather unnecessary if we use MetaMask signing. So we don’t recommend to funnel all users through Vault.

The value of Vault for the user is to have an independent view of their assets, as well as a tool to manage them, see history, etc. E.g. a user might use Vault when the dapp they interacted with has a faulty front-end.

The value of Vault for the developer is that they can delegate complex interactions like bridging and account creation to the Vault. But there’s nothing which Vault can do but a dapp cannot do directly. It’s just a matter of convenience.

What can Vault do:

1. Create FT4 account in the dapp (if dapp supports account creation in Vault - they might not)
2. For a particular dapp, view assets, history.
3. Manage auth descriptors (e.g. revoke access)
4. Do transfers within a dapp
5. Do cross-chain transfers between dapps
6. Do bridge transfers
7. Combined bridge + cross-chain
8. CHR staking (???)

Wallet SDK

By Wallet SDK we mean a combination of FT4 Rell and TypeScript libraries with an example of a wallet front-end. It's supposed to inform devs of best practices.

Dapps and chains

What is a dapp?

It is a thing which is usually embodied as a collection of software components, which does something useful (for the user) and has some of its parts decentralized in some way.

Usually it has three parts:

- Front-end, i.e. user-facing parts
- Non-blockchain backend
- Decentralized (blockchain) backend

Only the decentralized backend is the essential part. It might be a smart contract running on a blockchain, or a blockchain, or several blockchains, or some other form of a distributed ledger, or a p2p protocol, …

Non-blockchain backend is pretty common in the Ethereum world, but frowned upon in Chromia dapps. Chromia gives you tools to avoid making a non-blockchain backend in most situations (we hope).

We often call the front-end part a "client". This comes from "client-server architecture" and three-tier architecture (web2) where the presentation tier is often called "client". Ethereum people might call their node implementation a "client", which is a bit weird. (In their view, node software is the client for the network.)

A dapp on Chromia would typically own at least one container and have at least one chain running in it.

By chain we mean a blockchain, i.e. a chain of blocks. Blocks contain transactions. Transactions contain operations. Operations describe what the user wants to do.

We generally recommend avoiding terms like sidechains or appchain. While chains running on Chromia can be described as sidechains or appchains, it's simpler to just call it chain/blockchain.

Network, nodes, etc.

Validators

A lot of blockchain people use the term "validator" to mean a node which is involved in block production, e.g. it would sign blocks. Presumably, it's because it's supposed to sign only valid blocks.

This is stupid. All nodes check block validity before applying them. Calling only some nodes "validators" might confuse people.

We call them block producers or signers.

Sometimes people call entities which run such nodes "validators". We call them providers.

Providers

Providers are entities which run Chromia nodes. We distinguish

- Dapp providers: control dapp deployments, but might not run any nodes (and nodes they run are gonna be their local, semi-private nodes)
- Node providers: normal providers, which provide nodes for dapps to run on.
- System providers: providers who are able to run system cluster nodes and vote on important decisions
- Community providers: members of community who stake CHR and run replica nodes (currently not implemented)

Network, cluster, node, subnode…

A node is a server which runs Chromia software.

The role of a node is to run blockchains, syncing them with other nodes. But on Chromia, not every node runs every blockchain.

Instead, nodes are grouped in clusters, and clusters run the same blockchains.

Clusters are organized into a network. A network has a single system cluster which contains system blockchains, particularly, a Directory chain. The Directory chain describes the whole network.

Network unites clusters and chains running on them. Chains can send each other messages. (Or, rather, pull messages from each other.) Information from the Directory chain is used to verify messages (ICMF) & transaction proofs (ICCF).

System blockchains run directly on the node. Application blockchains run within a container, which is also called a subnode.

System chains

System chains are needed for the functioning of the network. System cluster contains:

- **Directory** chain - describes configuration of the network, holds dapp chain configurations (code), tells nodes which chains to run, etc.
- **Economic** chain
    - Origin of CHR token, CHR token minter
    - CHR EVM bridge
    - Staking
    - Staking rewards, provider rewards
    - Container leases
    - ??? cluster formation ???
- **System anchoring** chain - establish whole-network confirmation finality
- **Cluster anchoring** chains - feeds data to system anchoring, facilitates global topic ICMF, availability, etc.
- ??? **TransactionSubmitter** chain - facilitates transactions sent to EVM

Mainnet

Which components are essential for the launch of mainnet?

- Core (Postchain + Rell)
- Postchain-client
- FT4 (back-end and front-end libs)
- EVM Bridge
- System chains

FT4, part 2

FT4 is a protocol which handles authorization and tokens.
FT stands for “flexible tokens”.

4 is the version. FT1 was implemented in the JS version of Postchain. FT2 was implemented as a Kotlin module. FT3 was implemented in Rell. FT4 is also in Rell, but better.

Essential concepts:

- Account: Something which can hold tokens. Has an ID (aka address). Also something which can act.
- Authorization descriptor: A structure which describes what proof is necessary to authorize an action from a particular account. E.g. “To transfer tokens from this account you need a signature from this pubkey”. An account can have multiple authorization descriptors: e.g. master, temporary, etc.
- Special account: Account which user cannot directly act upon, which can be:
  - Global “smart contract” accounts.
  - Accounts which hold user’s locked tokens.

Are these tokens fungible? Yes. If you want them to. If you mint only 1 token, there’s nothing to “funge” it with, so it’s an NFT. So yes, FT4 can do NFTs just as well as fungible tokens.

FT4 has direct support for cross-chain transfers within Chromia using ICCF proofs.

There’s also a EVM asset bridge for FT4.

Originals/DIP

Originals is an asset description protocol (library): it describes what assets (tokens) represent. DIP is a highly flexible data storage & retrieval library which Originals makes use of.

DIP is “Dynamic Instance Protocol”, it uses entity–attribute–value model to store structured data without a fixed schema. (This is somewhat related to SemanticWeb, Resource Description Framework and triplestore, except that we do not implement any of w3c standards, functionally it’s very similar.)

Originals library is meant to be used together with FT4. FT4 says who owns the token. Originals say what the token is.

Note that besides the artwork, many kinds of data can be associated with a token, e.g.:

- Who minted it?
- Does it have any utility?
- What rights does the holder have?
- How can it be used in the game?

Originals can also describe the tokens which live outside of FT4, e.g. in EVM. That part is called **Metalayer**. (I.e. metadata storage layer.)

(Why is it called Originals? The main purpose is to describe original digital objects which are represented as tokens.)

Staking/rewards

Short version

Staking

The role of staking it is to make sure providers are economically aligned with the network and thus are unlikely to take actions which would harm it. It comes in two flavors:

- Providers **own stake**
- **Delegation** of stake from individual users

Users delegate their stake to providers (essentially, casting a vote of confidence) to get a share of revenue. Part of this revenue is a subsidy (i.e. part of a token supply), part comes from hosting fees.

Hosting fees

Application blockchains run in **containers**, which are carried by a group of nodes called a “cluster”. Dapps have to pay the network for containers allocated to them (i.e. once they **lease**). The amount of fees are determined by the size of the container (measured in Standard Container Units which include certain amount of compute, memory and storage space), extra storage, number of nodes in the cluster and extra features cluster might have (e.g. EIF - interop with EVM).

Fees are paid in CHR. Fees might be paid by the developer or by the dapp itself. Dapp might collect funds from users to pay the fees (e.g. charging for each transaction, subscription, etc).

Rewards

Providers receive rewards from the network for running the nodes.
The reward amount depends on factors such as

- Number and size of nodes which provider has
- The stake amount (both delegated and own)
- Quality of service (number of blocks proposed, etc)

Rewards come from **pools** which accumulate hosting fees and combine them with subsidy. Subsidy is meant to help the system when it’s new, but gradually it will be phased out, as there’s an upper cap on CHR supply, so CHR needed for subsidy cannot be minted forever.

Governance

Both reward and hosting fee models depend on various parameters which balance contribution of different factors. Optimal values for these parameters might change as Chromia evolves. Values of parameters can be updated by providers through voting, so that a supermajority is required to change a value. In future, some parameters might be updated automatically according to market situation, e.g. based on supply and demand.

Q&A

Is there slashing?

Some networks have slashing as means to penalize bad behavior.
The first version of Chromia mainnet will have a limited form of slashing - rewards can be withheld by quality-of-service factors (e.g. for nodes being down), but the staked amount stays constant.
We believe that depreciation of CHR locked in stake is sufficient to discourage bad behavior.
Future versions of Chromia might include slashing of the provider’s own stake.

How are tokens locked for staking?

Currently tokens are locked for two weeks. We believe this is enough to penalize harmful behaviors, as traders would have an ample time to react to changes before stakers & providers can cash out. We are considering longer lock times for providers’ own stake.

Why are tokens staked on Ethereum/BSC?

We believe this is convenient for users and provides an additional layer of resilience as stakers might vote on Ethereum chain in case of a major disruption of Chromia. Users can also stake tokens directly on Chromia.

How are providers selected?

Existing providers can add new providers to the network, presumably after they pass background checks (e.g. we want a provider to be a legitimate business and not a fake company for Sybil resistance reasons). But providers also need to satisfy stake requirements - e.g. have their stake and/or stake delegated to them by users.

Bridge & L2

What is L2?

L2 can mean many things, but in the context of Ethereum it is a blockchain which

1. Fully depends on another chain (L1) for security
2. Is almost as secure as L1, in the sense that users have a way to migrate to L1 if somebody bad happens to their L2
3. Has well-functioning bidirectional bridge bound by properties 1 and 2

It can also be less than a blockchain, say, a state channel, although modern discourse focuses more on chain-like things.

Given that people mostly care about tokens, L2 is essentially a bridge which is almost as secure as L1.

Entity which produces L2 blocks is usually called a sequencer (as their role is to sequence transactions into blocks), they can be multi-party.

Of course, it relies on assumptions. Nothing is secure unconditionally.

There are two kinds of commonly used L2 designs:

- Optimistic: Block summaries are posted on L1 by current validators. If somebody detects fraud they might submit fraud proof to take over (as well as canceling fraudulent transfers).
- ZKP: A zero-knowledge proof of state transition validity is posted on L1. Fraudulent transfer would have made such a proof impossible.

Data availability is important due to potential block withholding attacks: if sequencer posts a summary for a block which does not exist, a user won't be able to recover as at that stage the state is undefined. There are three main strategies to deal with data availability:

1. Post all block data needed to recover the state to L1. This is known as rollups.
2. A trusted party of a consortium promises to keep data safe.
3. A specialized "data availability" chain provides a proof. (Which is a proof that data was available at one time).

Is Chromia L2?

Not in the sense described above, but we might implement a proper Chromia-based L2 later

Hardened Bridge

Hardened bridge, or H-bridge is an implementation of a bridge between EVM chains and Chromia which is almost as secure as a proper L2, but does not go all the way.

It is inspired by an older L2 design called Plasma.

In the calm state withdrawal proofs are signed by Chromia nodes via block headers. This part is similar to a multi-sig bridge where signers are blockchain nodes.

An interesting thing happens when the majority of Chromia nodes are compromised. There are no fraud proofs, but honest nodes can trigger a mass-exit scenario. In mass-exit users can retrieve their balance based on a previously committed state. This is somewhat similar to Plasma.

It’s much harder for the attacker to profit from a compromised system, hence the bridge is called hardened.

Why is it not L2?

- There’s no guarantee that mass-exit is triggered
- Some number of transfers between last-known-good block and the fraudulent block might be lost

“The bridge is not the bridge”

There isn’t a single component which is called “the bridge”. Rather, it’s a combination of multiple components:

- EIF - Ethereum Interop Facility which itself consists of multiple parts
    - Import events from EVM
    - Collect events from Rell code organizing them into a Merkle tree to be used for proofs
    - Collect state trees
- Smart contracts on Ethereum - verify proofs, hold tokens, etc. This includes a library which can verify Chromia block headers in Ethereum
- FT4 connector
- (in progress) Transaction Submitter - send maintenance & anchoring transactions to EVM
- (in progress) anomaly monitoring - detect anomaly, trigger mass exit
- (in progress) mass exit

Our facilities

This is on the verge of being hardcore

EIF - Ethereum Interop Facility

- Imports events from contracts in EVM. This is relevant to the bridge but can also be used for things other than bridge, it can literally import any events.
- Creates event proofs which can be used on EVM to prove that things happened on Chromia. Again, relevant for the bridge but can be used for other things as well.
- TransactionSubmitter can submit transactions to EVM

ICMF - Inter-chain Messaging Facility

- One chain can publish a message, another can import it.
- Messages are verified automatically (obviously)
- Global topic can be used to collect messages from many chains into one (perhaps, a system chain)
- ICMF is used by the system itself, e.g. to coordinate blockchain updates across the network.
- It can also be used for multi-chain applications

ICCF - Inter-chain Confirmation Facility

Allows one chain on Chromia to verify that a particular transaction was included in another chain on Chromia. This is used for cross-chain token transfers within Chromia. (And can be used to transport arbitrary events.)
Difference from ICMF: ICMF messages are carried out by blockchain processes themselves, ICCF-based messaging are carried by user messages.

Confirmations, anchoring, etc.

What does it mean for the transaction to be confirmed?

Suppose Alice is selling Bitcoins for cash. Bob will give cash only when he is sure that he'll get Bitcoins, i.e. Alice's transaction cannot be reversed or modified. This is also known as finality. This is something users (particularly, merchants) seek.
But Bitcoin can only provide probabilistic confirmations. If Bob waits e.g. 10 blocks to be built on top of the block which includes Alice's transaction, it's very likely that it won't be reversed. But you can never be 100% sure.

In practice, the above is true only if Bob runs a node itself and it is connected to many other honest nodes. If Bob uses a mobile wallet which is not a light node, he's essentially at mercy of the server that mobile wallet connects to.

On Chromia, each blockchain has N signers (block producers). Signatures of 67% of these nodes is sufficient to prove that a block was committed. A committed block cannot be removed or modified unless more than 33% of nodes conspire to split the chain. 33% is sufficient only in an ideal situation. In practice, 67% of nodes would be needed to maintain the split.

Thus, on Chromia we have 1-block finality as long as you believe it's extremely unlikely that a lot of nodes will be compromised.

However, if you're an exchange working with large sums of money, you might seek additional verification which works even if 100% nodes are compromised.

Anchoring

Anchoring is a process of including block headers (or, at least, their hashes) of one blockchain into another (perhaps more secure) blockchain.

E.g. suppose you're transacting on chain C and you're concerned that it might be compromised - perhaps, the blocks you're seeing are different from ones which others see.

Suppose there's another chain A which you trust more. If block header C_1001 is included in chain A, you might be more certain that C_1001 is the only C's block at height 1001, i.e. there are no two versions of C. We say that C is anchored in A.

Thus if you received a payment in block C_1001 you might want to wait until it shows up in A, i.e. your definition of "confirmed" includes "confirmed in C and anchored into A".

In Chromia, we use hierarchical anchoring. Dapp chain blocks are anchored in "cluster anchoring chain", one per cluster. (This does not add security but increases scalability.)
All cluster anchoring chains get anchored into system anchoring chain, maintained by the system cluster.
System anchoring chain is itself anchored in Ethereum (and, perhaps, at some point, in Bitcoin).

Upgrades, configurations, forks

In other blockchains

Different protocols use different upgrade methods.

Bitcoin is highly conservative, its developers prefer a mechanism called “soft fork”: rules can only be made more restrictive. Old versions of Bitcoin node software will accept blocks produced by updated nodes. New functionality can be added, as, e.g., previously ignored opcodes might mean something for new nodes. But data formats can’t be changed, etc.

Ethereum is less conservative, and is updated using “hard forks”: at a particular time new rules apply. This requires all node operators to update at a particular time. (Or, if they disagree, they might never update.)

If a fork is contentious or buggy, it might result in a chain-split situation - two different versions of a chain exist at the same time.

Smart contracts on Ethereum can sometimes be updated using rules defined in smart contracts. Perhaps, multi-sig authorization. Sometimes this results in contracts being compromised.

In Chromia

Chromia has a built-in mechanism to update any chain, including system chains.

A particular version of blockchain’s code is called a *configuration*. New configurations can be submitted to the Directory chain and voted on. If they are approved by a voting set of a particular container a blockchain is hosted on, nodes will try to switch to the new configuration.

A configuration can also be submitted by a blockchain itself, in this case updates can be controlled by a DAO (users). (Note: Not implemented yet.)

Chromia also includes a mechanism to create a fork of a blockchain, i.e. a copy of blockchain data which has its own unique ID and might use different code to process the same data.

Forks can be used for many things:

- Fixing a bug which “corrupted” the state - this would require consent of users (to switch to new ID)
    - This might screw cross-chain bridged tokens
- Create a clone of something
- Apply additional processing to data from an existing blockchain

---

Hardcore topics

Special transactions

In blockchains like Ethereum all transactions have to be initiated externally. I.e. a blockchain can only do something only in a response to a transaction.

On Chromia we have *special* transactions which are added to the blockchain by nodes themselves. For practical purposes, it's just one transaction added at the beginning of a block, but it might include many operations. So, rather, we can talk about special operations.

Special transactions do not have a signature, but they are validated according to custom rules dependent on operation type.

Currently following operations are used in Chromia:

- Application-defined code called at the beginning of the block, can be used e.g. to perform periodic actions, cleanup, etc.
- ICMF & anchoring is implemented using special operations.
- EIF imports events from EVM using special operations.

More things can be added in the future. E.g. special transactions can interact with external systems, run off-chain computations, etc.