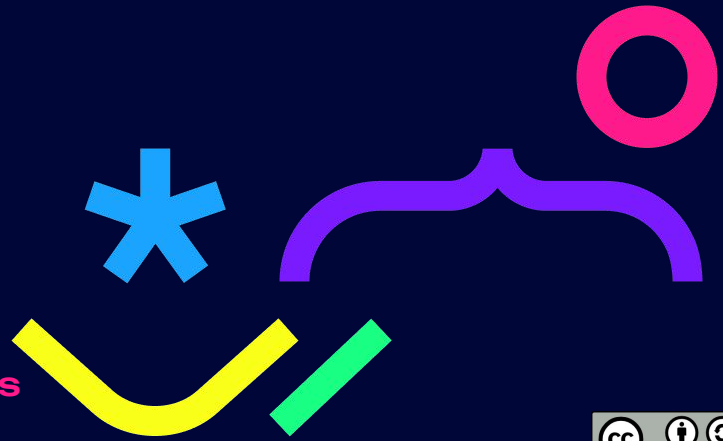
Commons

XRP Ledger Blockchain Day

TUM Blockchain Conference

September 2025



Dr. Vera Radeva, Head of Education @ XRPL Commons
Mr. Mathis Sergent, Research Engineer @ XRPL Commons

You are holding open source content.

Here's how to handle it:

→ You can use this content in your work, adapt, and share it.

→ You must mention XRPL Commons and the Creative Commons license as a source.

Creative Commons Attribution-ShareAlike CC BY-SA – This license allows you to protect, reuse and adapt this content even for commercial purposes, if you mention XRPL Commons as a source and allow your re-adapted content to be under the same Open Source License.

Commons

Our mission is to create the conditions of success for builders, entrepreneurs and developers, to thrive in the XRPL ecosystem.

🎓 Learn | 💻 Build | 🌊 Launch | 🤝 Adopt | 🫶 Engage

XPRL Commons

Learn

The learning curve for Blockchain is steep. It's better to be supported whether you have tech or business roles.

Build

Our in-house residency and hackathons offer builders a playground for invention.

Launch

Accelerate your project at the Aquarium Residency, where developers and entrepreneurs transform ideas into reality.

Adopt

Blockchain integration for corporations and large institutions with tailored support and resources.

Engage

Meetups, podcasts, our Community Magazine and news on our socials unite the XRPL ecosystem.

Learn

Teach

Democratize access to knowledge on blockchain technology

chaise
blockchain skills for Europe

Research

Support research development of the XRP Ledger

ripple

Chaire UNESCO ITEN Innovation, Transmission, Edition Numériques

ILB Institut Louis Bachelier

isae SUPAERO Institut Supérieur de l'Aéronautique et de l'Espace

Network

Campus Ambassador Program

KRYPTOSPHERE®

TUM BLOCKCHAIN CLUB

2 year achievements

1000+
Students **reached**

14+
University **partnerships**

65+
Educational videos available on **our Youtube channel**

280+
People **benefited** from HQ trainings

42+
New hands-on **blockchain and XRPL courses**

3+
Research labs visitsd one research intern

10+
University **partnerships**

Educational Playground: OASIS

🌴 This platform is designed to **empower you** with the latest **knowledge in blockchain** technology and to help you **enhance your teaching skills**

Educational Playground: OASIS

🌴 This platform is designed to **empower you** with the latest **knowledge in blockchain** technology and to help you **enhance your teaching skills**

Find high-quality **educational resources** on blockchain

Receive **training in blockchain** technology

Acquire **teaching opportunities** & identify potential Web3 partners

Integrate blockchain technology into your **academic curriculum**

Develop a new blockchain course that meets **market needs**

Core Dev Bootcamp

Learn the core of the XRPL protocol intensively, in person, over 2 weeks.

- How to develop new features on the XRPL.
- Designed to be efficient with monthly disbursements
- The last session welcomed 17 people from 8 different nationalities.

Doctor Block: The Thesis Challenge

- **I Round:** Apply and submit 3 min video on the topic of your PhD thesis before 31.03.2026
- **II Round:** In person pitch in Paris with pre-selected candidates in front of a jury in June 2026
- **Prizes:**
  - Be featured at the Community Magazine
  - Participate in the creation of a new blockchain education manual
  - Become a member of the XRPL Commons Teachers Pool

DoctorBlock
The Thesis Challenge

🎓 The **Campus Ambassador Program** aims to actively support **student blockchain clubs** across the world

Our partners

Kryptosphere
20+ campus, **3** countries (FR, UK, BE)
600+ students

TUM Blockchain Club
53+ members (GY)
2,000+ students

Forum Trium
200+ businesses, **4+** schools
4,000+ students

DE CIFIRIS
300+ members, teachers and students (IT)

What's next?

Trainings

TUM Blockchain	11–12 September 2025
Léonard-de-Vinci	19 - 20 September 2025
Ecole 42	29 - 30 September 2025

Hackathons

IXH24	6–8 November 2025

Conferences

KRYPTO-TOUR	11 October 2025
Cryptography and Blockchain	14 October 2025

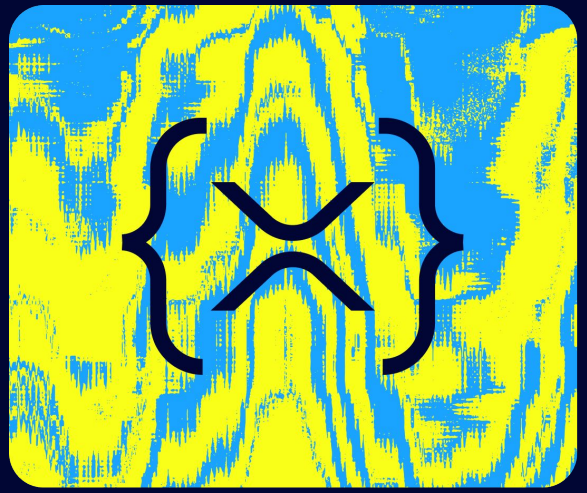
Careers Forum

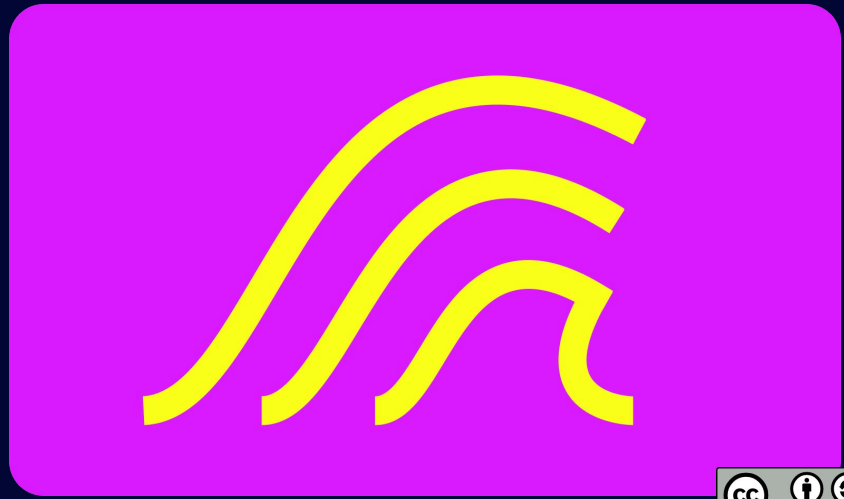
Université Paris Cité	November 2025

The Aquarium 🌊

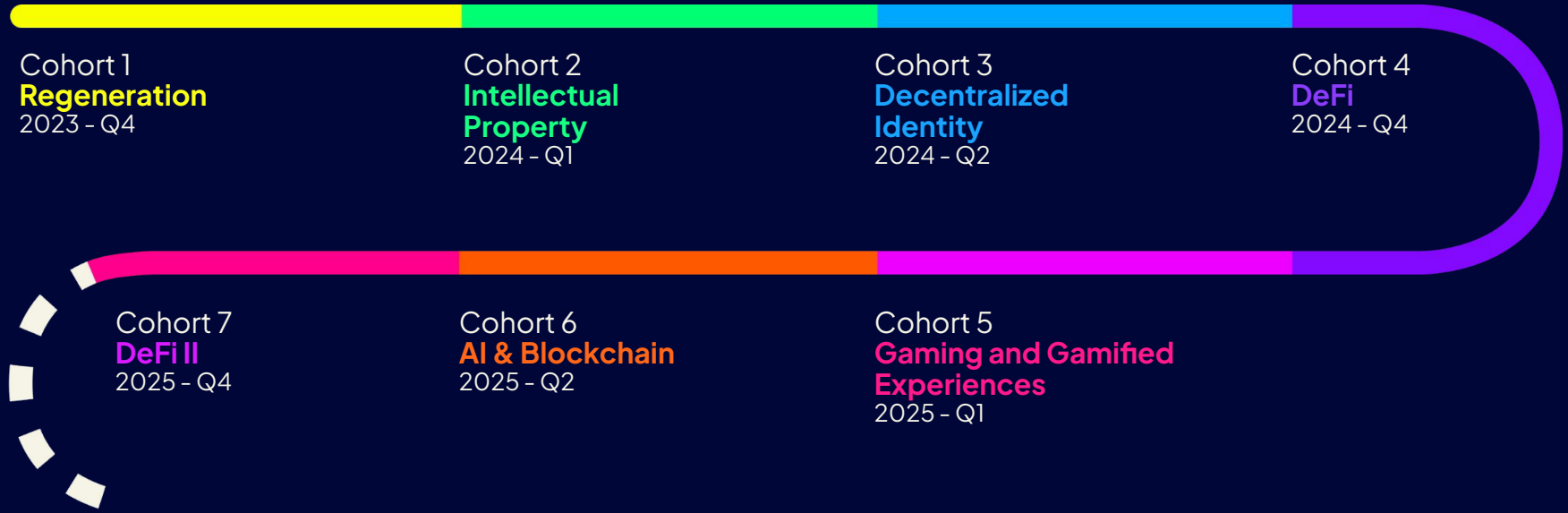
12 weeks to build impactful projects on the XRP Ledger

March 2025





Our cohorts focus on key emerging real-world blockchain applications



People → Projects → Companies

All our residents

Cohort 1

Cohort 3

Cohort 5

Cohort 6

Cohort 2

Cohort 4

Join us!

Get featured in our Community Magazine

Introduce us to your Lab & PhDs students

Become a mentor to entrepreneurs

Join our pool of instructors

Record a 5 min video on your favorite blockchain concept

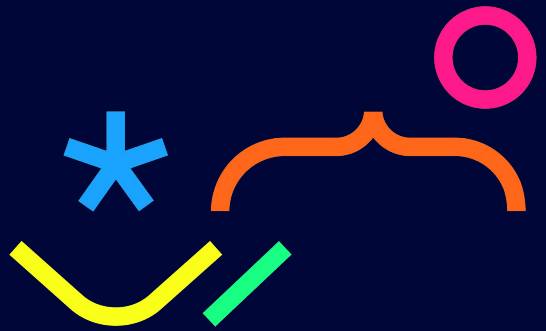
Come or send your students to our free dev. trainings in Paris

Contribute to OASIS content creation

www.xrpl-commons.org

Tag & Follow XRPL Commons

- Published a post on socials?
  Great, please tag us so we can share it as well!
- **@XRPL_Commons on all platforms**



Let's connect!

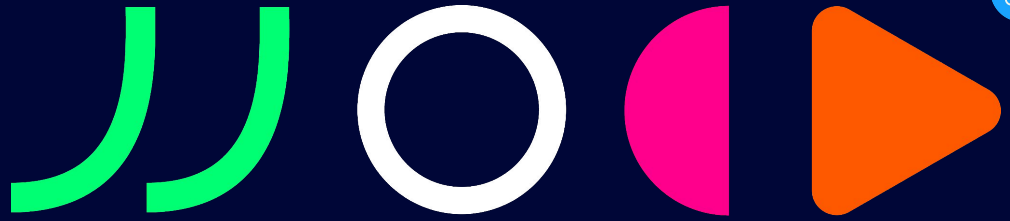

Let's Build!

Let's dive into the code

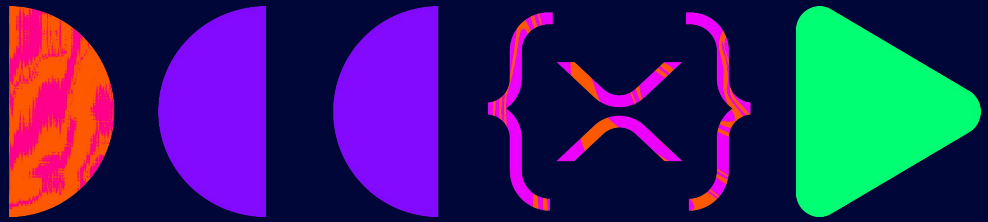
- 💙 Presentation of XRPL Commons
- {X} Introduction to the XRP Ledger
- 🔀 Wallet & Transactions
- 🌐 Multi-signature Transactions
- 🆕 Batch Transactions
- ON! Issued Tokens & TrustLines
- 💱 Decentralized Exchanges (DEX): AMM & CLOB

Let's dive into the code

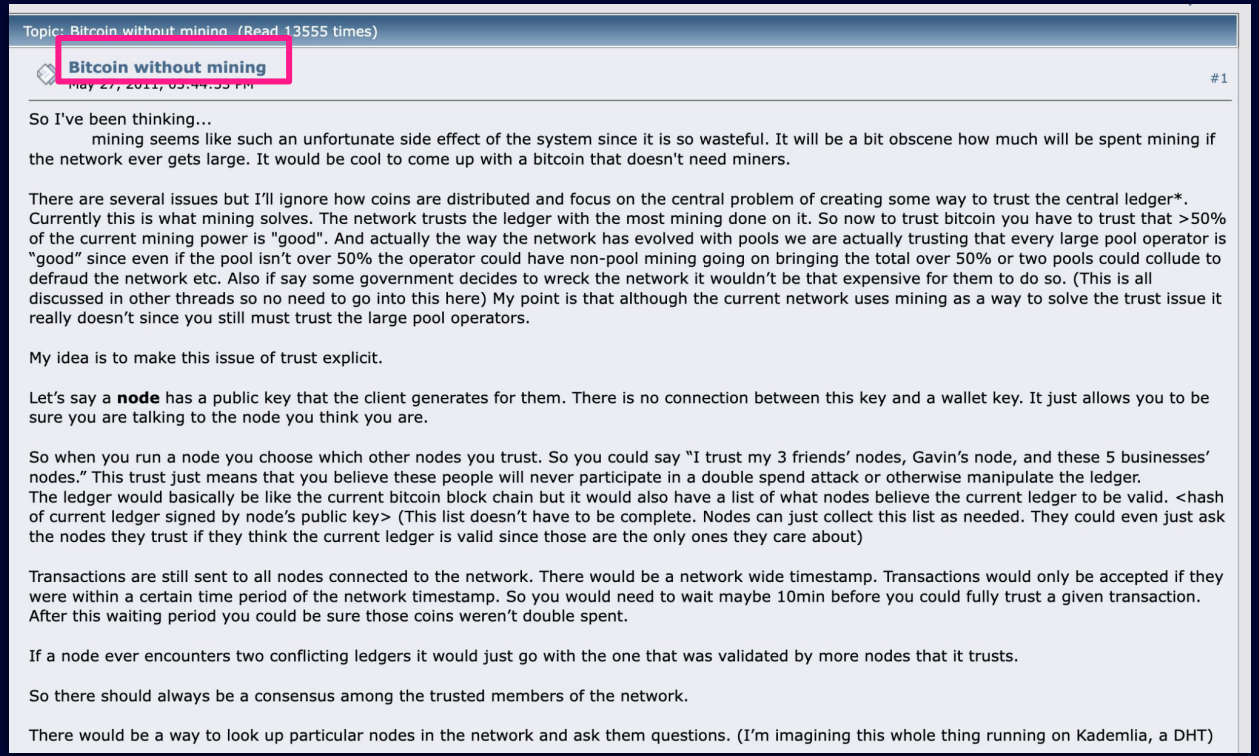
github.com/XRPL-Commons/xrpl-day-tum-2025



Introduction to the XRP Ledger



Bitcoin's Limitations



Topic: Bitcoin without mining (Read 13555 times)

Bitcoin without mining

May 27, 2011, 03:44:55 PM #1

So I've been thinking...
mining seems like such an unfortunate side effect of the system since it is so wasteful. It will be a bit obscene how much will be spent mining if the network ever gets large. It would be cool to come up with a bitcoin that doesn't need miners.

There are several issues but I'll ignore how coins are distributed and focus on the central problem of creating some way to trust the central ledger*. Currently this is what mining solves. The network trusts the ledger with the most mining done on it. So now to trust bitcoin you have to trust that >50% of the current mining power is "good". And actually the way the network has evolved with pools we are actually trusting that every large pool operator is "good" since even if the pool isn't over 50% the operator could have non-pool mining going on bringing the total over 50% or two pools could collude to defraud the network etc. Also if say some government decides to wreck the network it wouldn't be that expensive for them to do so. (This is all discussed in other threads so no need to go into this here) My point is that although the current network uses mining as a way to solve the trust issue it really doesn't since you still must trust the large pool operators.

My idea is to make this issue of trust explicit.

Let's say a **node** has a public key that the client generates for them. There is no connection between this key and a wallet key. It just allows you to be sure you are talking to the node you think you are.

So when you run a node you choose which other nodes you trust. So you could say "I trust my 3 friends' nodes, Gavin's node, and these 5 businesses' nodes." This trust just means that you believe these people will never participate in a double spend attack or otherwise manipulate the ledger.
The ledger would basically be like the current bitcoin block chain but it would also have a list of what nodes believe the current ledger to be valid. <hash of current ledger signed by node's public key> (This list doesn't have to be complete. Nodes can just collect this list as needed. They could even just ask the nodes they trust if they think the current ledger is valid since those are the only ones they care about)

Transactions are still sent to all nodes connected to the network. There would be a network wide timestamp. Transactions would only be accepted if they were within a certain time period of the network timestamp. So you would need to wait maybe 10min before you could fully trust a given transaction. After this waiting period you could be sure those coins weren't double spent.

If a node ever encounters two conflicting ledgers it would just go with the one that was validated by more nodes that it trusts.

So there should always be a consensus among the trusted members of the network.

There would be a way to look up particular nodes in the network and ask them questions. (I'm imagining this whole thing running on Kademlia, a DHT)

Source: https://bitcointalk.org/index.php?topic=10193.0

XRPL was created in 2012

1991- 2000
Publication of work on cryptography & Merkle trees

2004
Hal Finney introduces a digital currency system to solve the double spending problem

2008-2009
Satoshi Nakamoto publishes a whitepaper on Distributed Blockchain. The first bitcoin block is mined by Satoshi

2012
Creation of the XRP Ledger

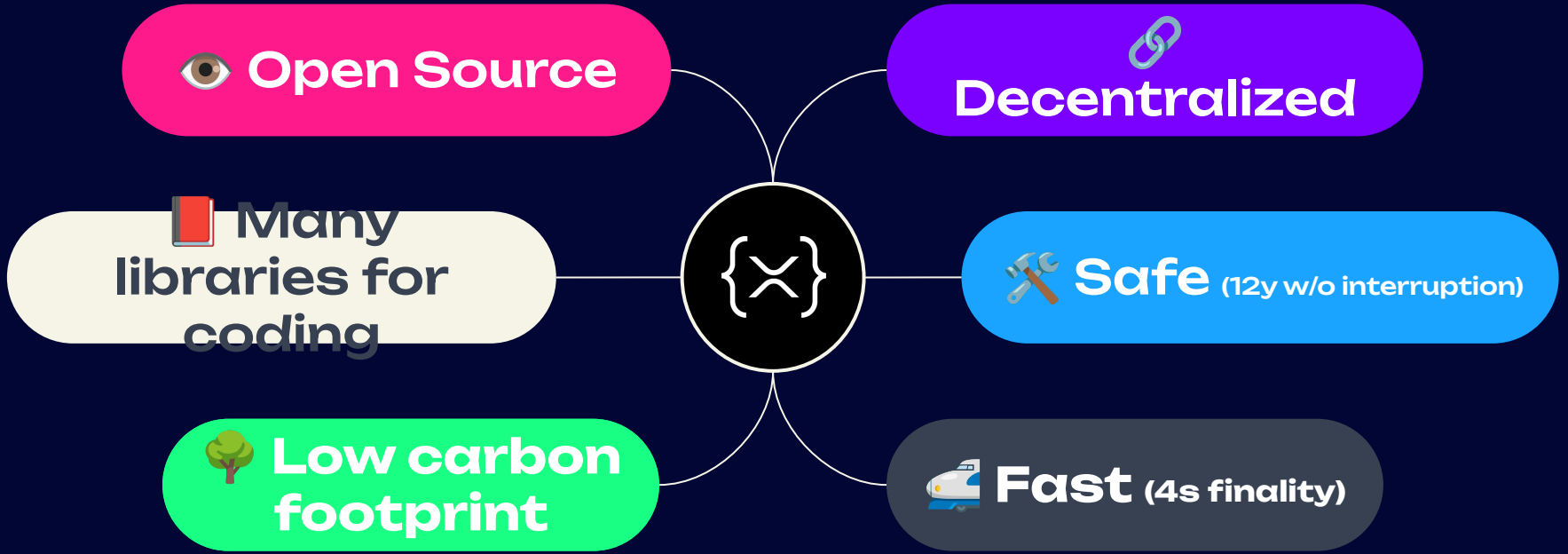
2014-2016
Vitalik Buterin creates the Ethereum network

2017 - 2021
Acceleration of DeFi, NFTs and Metaverse

2024-onwards
Bull market, Bitcoin and other cryptos reach all time high

Commons

The XRP Ledger



The differences between the XRP Ledger, XRP, and Ripple

XRP LEDGER

Layer-1 blockchain

The XRP Ledger is a decentralized public blockchain that is open-source and powered by a global developer community.

Companies, institutions, developers and individuals around the world use the XRP Ledger for use cases across, tokenization, payments, stablecoins, CBDCs, and more.

XRP

Native Digital Asset

XRP is the native digital asset (token) of XRPL, similar to ETH for Ethereum or SOL for Solana.

The primary utility of XRP is to facilitate transactions on the network.

XRP also serves to protect the ledger from spam, and to bridge currencies in the XRP Ledger's DEX.

ripple

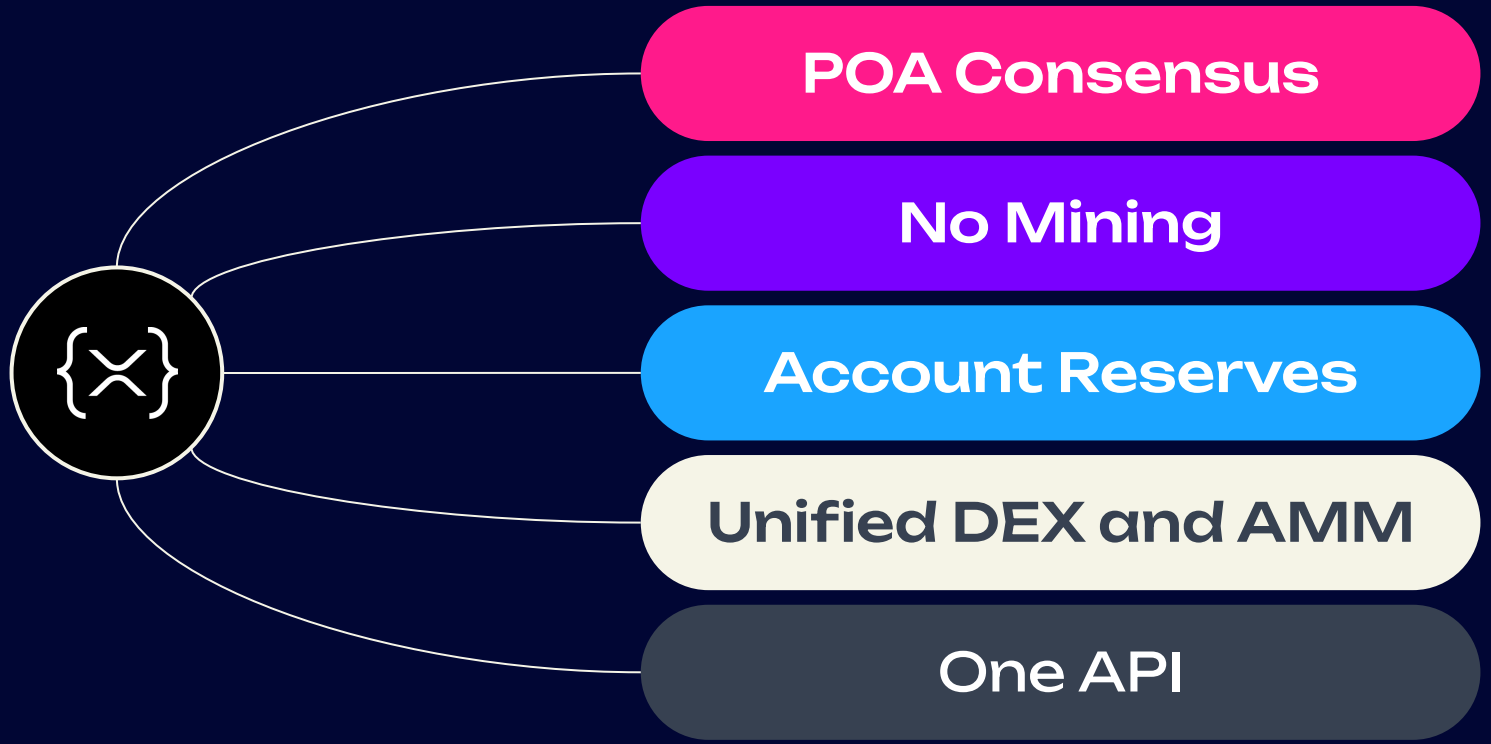
Crypto Solutions Company

Ripple is a technology company that builds crypto solutions for businesses. Ripple uses XRP as a bridge currency in its solutions because it's fast, efficient, and reliable, making it ideal for financial use cases like payments. Ripple is also a holder of XRP.

Ripple is one of many companies building on and contributing to the XRP Ledger.

Source: https://ripple.com/xrp/, https://xrpl.org/faq.htm, https://ripple.com/insights/q1-2023-xrp-markets-report/l

Specificities of the XRPL



XRPL Consensus

The XRPL consensus is a type of **Proof of Association**

Explicit trust
Hundreds of validator nodes participate in the consensus. **35 special nodes** are on the UNL which lists the nodes who have final say. 80% of the UNL must agree to validate a block.

Decentralized
No single entity can control more than 5% of the UNL. Every member of the UNL is a known entity with full transparency.

Independently governed
The XRPL foundation ensures UNL members adhere to strict guidelines of maintenance upgrades and uptime. UNL members are regularly audited and can change over time.

One Endpoint to do all the things

Single API

No need to stitch together disparate systems or spend months integrating complex technology - simply connect into XRPL through a single API

Minimal Code Required

Astonishingly simple, you can get up and running on the XRP Ledger in as little as few lines of code using familiar programming languages (JS, Python, Java, and many more)

Powerful built-in protocol features

Native DEX

First on-chain DEX in the world, trading and moving tokens anywhere in seconds with competitive liquidity

Issued Assets

Ability to represent digital currencies, legal obligations, fungible tokens, and other asset classes on the ledger

Non-Fungible Tokens

Implements non-fungible tokens with built-in royalties where all trades handled by the DEX

Token Asset Controls

Controls for token issuers and holders to enhance security and regulatory compliance

Advanced Payments

Use advanced payment capabilities like “Escrow” and “Checks” to build smart applications without smart contracts

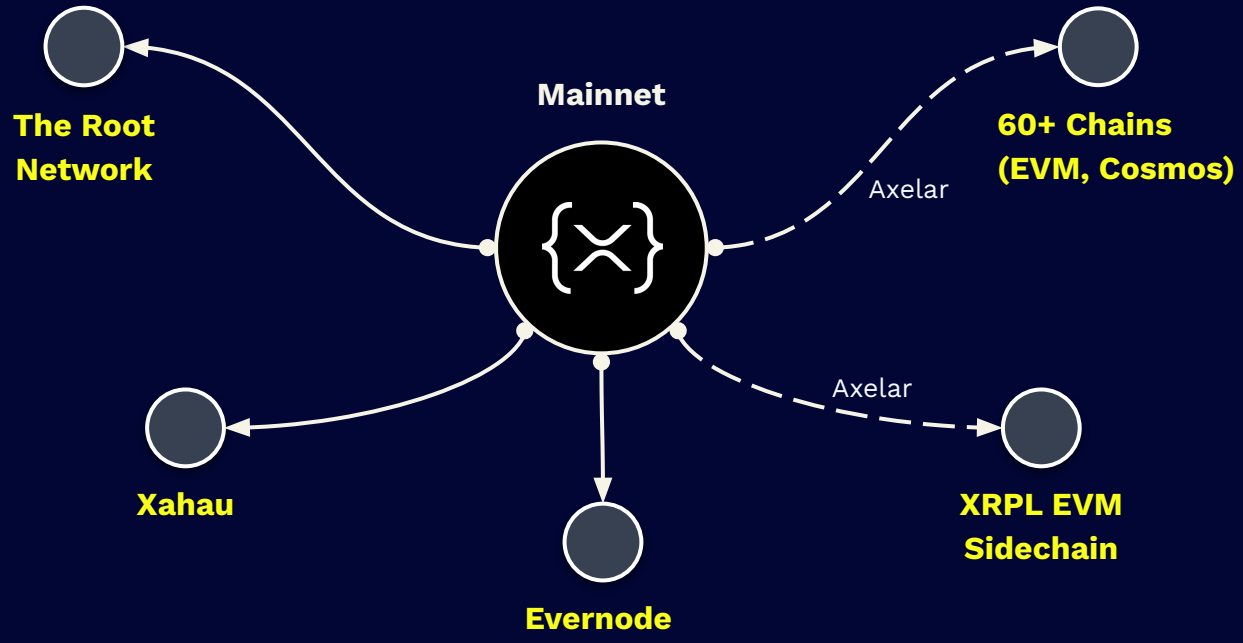
Automated Market Maker

Liquidity pools bring yielding assets to the ledger as well as the ability to provide liquidity on your tokens

https://xrpl.org/configure-rippled.html

The XRPL Extended Ecosystem

XRPL Mainnet interoperates with sidechains, to bridge XRP and tokens



The XRP Ledger has two core server modes

Stock Server

Multipurpose configuration to **submit transactions to the XRPL, access ledger history** and use the latest **tools to integrate** with XRP and the XRP Ledger.

Validator

Does everything a stock server does except that validators **broadcast signed transactions**, which are essential to the XRPL **consensus mechanism**

Each server can be configured across several capabilities

Data retention

Control how much data your server should keep and when it should remove old data (i.e. full history, online/advisory deletion, history sharding)

Peering

Configure how your server connects to the peer-to-peer network (i.e. clustering servers, reserved peer spots or blocking unwanted peers,..)

Amendment Voting

Set your validator server's votes on protocol amendments

StatsD

Monitor and visualize your rippled server with StatsD health and behavioral metrics

Parallel Networks

Connect your rippled server to the Testnet or Devnet to try out new features or test functionality with fake money

gRPC (Remote Proc. Call)

Enable and configure the gRPC API, allowing computer programs to execute in a different address space in which is written

Source: https://xrpl.org/configure-rippled.html

🔎 Data Retention

Control **how much data** your server should keep and when it should **remove old data**

Full History
Provides a record of every transaction ever to occur in the XRP Ledger, although they are expensive to run

Online deletion
Purges outdated transaction and state history. Can be done based on how far back to store transaction history or deleting holder ledger history on a schedule

History Sharding
Divides the work of keeping historical ledger data among rippled servers

Source: https://xrpl.org/configure-rippled.html

XRPL has matured into one of the most robust layer-1 blockchains

100%

decentralized blockchain with 600+ nodes processing transactions and maintaining the ledger

1,750+

unique apps and exchanges on mainnet built by a diverse set of global developers

5M+

active XRP wallet holders around the world

185+

Proof-of-Association validators operated by universities, exchanges, businesses, & individuals

2.8B+

transactions processed representing over $1T in value moved between counterparties

~$125B+

market capitalization of XRP, making it the ~5th largest cryptocurrency

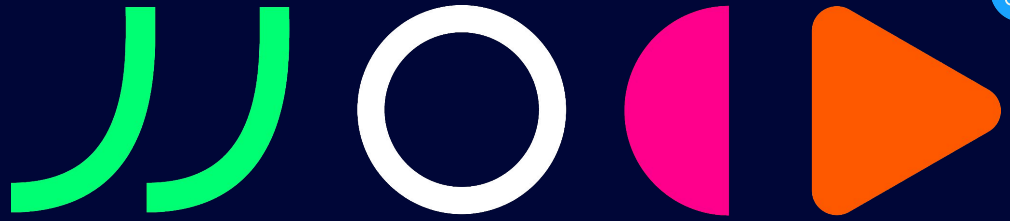
Source: CoinMarketCap, May 2025 https://livenet.xrpl.org/

Addresses on the XRP Ledger

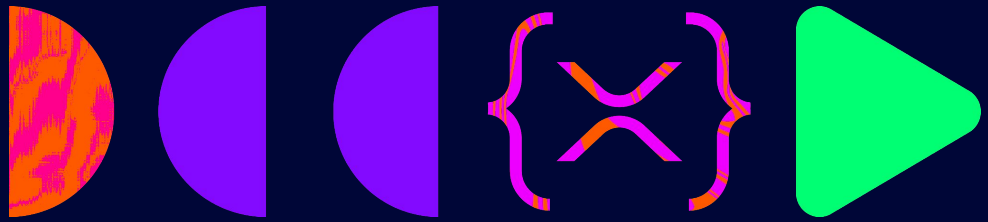
Data type	Start with	Type prefix	Maximum number of characters
Account address	r	0×00	35
Account public key	a	0×23	53
Node public key	n	0×1C	53
Private key	s	0×21	29

Assess your understanding 🧠

Quiz!



Wallets & Transactions 🔀



👨‍💻 Resources for developers

⬅️ https://xrpl.org/docs

https://docs.xrpl-commons.org ➡️

⬅️ https://github.com/XRPLF/xrpl-dev-portal/tree/master/_code-samples

Basics of Transaction on the XRPL

Each transaction has the same set of **Common Fields** plus **Additional Fields** based on the **Transaction Type**

Some Common Fields:

- **Account**: The address initiating the transaction
- **TransactionType**: Defines the transaction type (Payment, OfferCreate, etc.)
- **Fee**: Amount of XRP to be destroyed as transaction cost
- **Sequence**: Transaction sequence number for the account

Transactions Types: Building Block of XRPL's Features

Transaction Types = **Specific operations** or actions on the XRPL
One feature may require multiple transaction types to function completely
Each transaction type has a specific purpose and schema

Examples:

NFT Feature includes multiple transaction types:

- ***NFTokenMint****: Creates a new NFT*
- ***NFTokenCreateOffer****: Lists an NFT for sale*
- ***NFTokenAcceptOffer****: Completes an NFT transfer*

Transaction types are the fundamental units of work on the ledger, and complex features are built by combining them.

Flags: Adding Customization to Transactions

Flags = Options that **modify a transaction's behavior**
Allow developers to customize transaction logic without requiring new transaction types
Can be combined (bitwise) to create complex behaviors

Examples:

NFTokenMint Flags:

- **tfTransferable**: When disabled (0), makes NFT non-transferable
- **tfBurnable**: When enabled (1), allows NFT to be burned by issuer

Flags allow fine-grained control over transaction behavior without requiring new transaction types for every variation.

General Process to send a Tx

1. Connect to a client
2. Get your wallet
3. Prepare & Sign the transaction
4. Get the result

1. Connect to a Node

```
import { Client } from "xrpl";

const client = new Client("wss://s.altnet.rippletest.net:51233");

async function main() {
  await client.connect();
  await client.disconnect();
}
```

2. Create a Wallet

```
// Generate key pair and call the faucet
async function createWallet(client: xrpl.Client) {
  const { wallet, balance } = await client.fundWallet();
}
```

3. Prepare & Send the Transaction

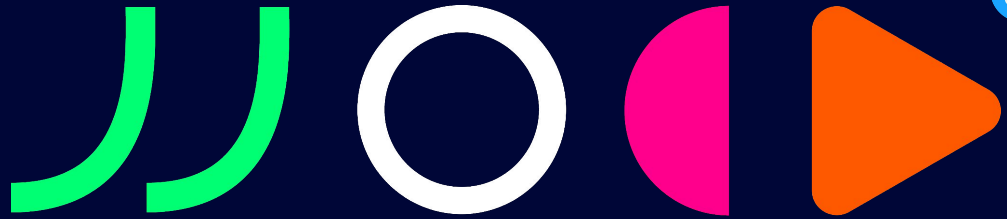
```
async function sendPaymentTx(
  client: xrpl.Client,
  wallet: xrpl.Wallet,
  address: string,
  amount: number,
) {
  const tx: xrpl.Payment = {
    TransactionType: "Payment",
    Account: wallet.classicAddress,
    Destination: address,
    Amount: xrpl.xrpToDrops(amount),
  };

  return await client.submitAndWait(tx, {
    autofill: true,
    wallet,
  });
}
```

4. Get the Result

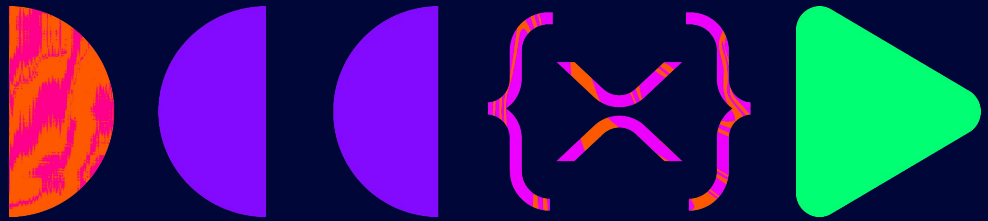

```
{
  api_version: 2,
  id: 22,
  result: {
    close_time_iso: "2024-12-06T13:39:10Z",
    ctid: "C02C0E9E00010001",
    hash: "A2D8910A45D19A91755F3BBC1E29F5FC97C438B0E117E3FBD042DAD1C9A94B98",
    ledger_hash: "8DAABD0B422F691F3E806EF0E78B8D3F5D7F7D831E97661754A6E31B5E7CEA7C",
    ledger_index: 2887326,
    meta: {
      AffectedNodes: [ { ModifiedNode: [Object] }, { CreatedNode: [Object] } ],
      TransactionIndex: 1,
      TransactionResult: "tesSUCCESS",
      delivered_amount: "10000000"
    },
    tx_json: {
      Account: "rELCjuVzgjBrBUexXoytR59gzEMTU3BwBc",
      DeliverMax: "10000000",
      Destination: "rf26gfMAfxaSK8cRJ8b3HpSn11N4v5xD9h",
      Fee: "12",
      Flags: 0,
      LastLedgerSequence: 2887344,
      Sequence: 2887324,
      SigningPubKey: "ED598042DFC6F9C65B16002F042B57AA4372EC6B12B9F3862F772A4FCF7B331BBC",
      TransactionType: "Payment",
      TxnSignature: "4ABCFD76572234A1B51BE6509C6364835530EE44C2B59005D4A5ACE4C84B13F9EF0025CF84A5C1F922F500877E2BCFA
0C6696A434024FD2E3A776E0F7B485C07",
      date: 786807550,
      ledger_index: 2887326
    },
    validated: true
  },
  type: "response"
}
```

The Entire Code

```
import xrpl from "xrpl";

const client = new xrpl.Client("wss://s.altnet.rippletest.net:51233");

async function main() {
  // 1. we connect to a node
  await client.connect();

  // 2. we create a wallet
  const wallet = await createWallet(client);

  // 3. we prepare and send the tx
  // Here we want to send 10 XRP to rf26gfMAfxaSK8cRJ8b3HpSn11N4v5xD9h
  const amount = 10;
  const tx = await sendPaymentTx(
    client,
    wallet,
    "rf26gfMAfxaSK8cRJ8b3HpSn11N4v5xD9h",
    amount,
  );

  //4. we get the result
  console.log(tx);
  await client.disconnect();
}
```



Multi-signatures 👨‍👩‍👧‍👦



1. Signers list configuration

```
console.log(chalk.bgWhite("-- SETUP SIGNERLISTSET --"));

const signerListSetTx = {
    TransactionType: "SignerListSet" as const,
    Account: multisig.classicAddress,
    SignerQuorum: 2,
    SignerEntries: [
        { SignerEntry: { Account: wallet1.classicAddress, SignerWeight: 1}},
        { SignerEntry: { Account: wallet2.classicAddress, SignerWeight: 1}}
    ],
};

const preparedSignerListSetTx = await client.autofill(signerListSetTx);
const resultSignerListSetTx = await client.submitAndWait(preparedSignerListSetTx, {
    wallet: multisig
});

if (resultSignerListSetTx.result.validated)
    console.log(`✅ SignerListSet created. Tx: ${resultSignerListSetTx.result.hash}\n`);
else
    console.log(chalk.red(`❌ Error creating SignerListSet: ${resultSignerListSetTx}\n`));
```

2. Create tickets

```
console.log(chalk.bgWhite("-- CREATE TICKETS --"));
const ticketCount = 5;

const createTicketTx = {
    TransactionType: "TicketCreate" as const,
    Account: multisig.classicAddress,
    TicketCount: ticketCount
}

const resultCreateTicketTx = await client.submitAndWait(createTicketTx, {
    autofill: true,
    wallet: multisig
});

if (resultCreateTicketTx.result.validated)
    console.log(`✅ Tickets created. Tx: ${resultCreateTicketTx.result.hash}\n`);
else
    console.log(chalk.red(`❌ Error creating tickets: ${resultCreateTicketTx}\n`));
```

3. Retrieve tickets

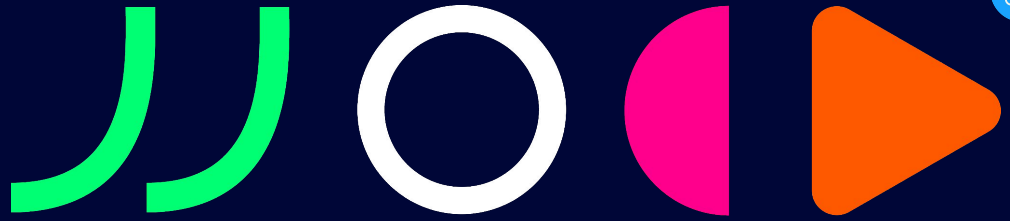
```
console.log(chalk.bgWhite("-- RETRIEVE TICKETS --"));

const accountTickets = await client.request({
    command: 'account_objects' as const,
    account: multisig.classicAddress,
    type: 'ticket' as const,
});

console.log(`Active tickets: ${JSON.stringify(accountTickets.result.account_objects, null, 2)}\n`);
```

4. Optional: Disable the master key

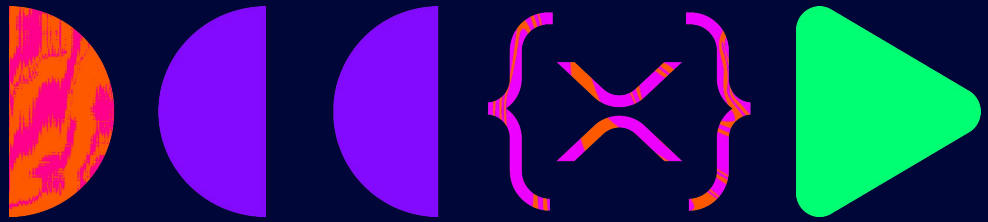
```
console.log(chalk.bgWhite("-- DISABLE MASTER KEY --"));
const accountSetTx = {
    TransactionType: "AccountSet" as const,
    Account: multisig.classicAddress,
    SetFlag: AccountSetAsfFlags.asfDisableMaster, // Disable master key
}
const resultAccountSetTx = await client.submitAndWait(accountSetTx, {
    autofill: true,
    wallet: multisig
});
if (resultAccountSetTx.result.validated)
    console.log(`✅🔒 Master key disabled. Tx: ${resultAccountSetTx.result.hash}\n`);
else
    console.log(chalk.red(`❌ Error disabling master key: ${resultAccountSetTx}\n`));
```

5. Execute a multisig transaction

```
    console.log(chalk.bgWhite("-- SEND MULTISIG TRANSACTION --"));

    const multisigPaymentTx: Payment = await client.autofill({
        TransactionType: "Payment",
        Account: multisig.classicAddress,
        Destination: wallet1.classicAddress,
        Amount: xrpToDrops(1)
    }, 2);

    const signedTx1 = wallet1.sign(multisigPaymentTx, true);
    const signedTx2 = wallet2.sign(multisigPaymentTx, true);

    const multisigedTx = multisign([signedTx1.tx_blob, signedTx2.tx_blob]);
    const resultMultisigTx = await client.submitAndWait(multisigedTx);

    if (resultMultisigTx.result.validated)
        console.log(`✅ Multisig transaction sent. Tx: ${resultMultisigTx.result.hash}\n`);
    else
        console.log(chalk.red(`❌ Error sending multisig transaction: ${resultMultisigTx}\n`));

    await client.disconnect();
};
```



Batch transactions 🆕



1. Retrieve sequence for each account

```
const accountInfo1 = await client.request({
    command: 'account_info',
    account: wallet1.address
});

const accountInfo2 = await client.request({
    command: 'account_info',
    account: wallet2.address
});

const accountInfo3 = await client.request({
    command: 'account_info',
    account: wallet3.address
});

const sequence1 = accountInfo1.result.account_data.Sequence;
const sequence2 = accountInfo2.result.account_data.Sequence;
const sequence3 = accountInfo3.result.account_data.Sequence;
```

2. Prepare the batch transaction

```
const batchTx: Batch = {
    TransactionType: "Batch",
    Account: wallet1.classicAddress,
    Flags: BatchFlags.tfAllOrNothing,
    Fee: "20000", // Higher than regular transactions
    RawTransactions: [
        {
            RawTransaction: {
                TransactionType: "Payment",
                Flags: GlobalFlags.tfInnerBatchTxn,
                Account: wallet1.address,
                Destination: wallet2.address,
                Amount: "2000",
                Sequence: sequence1 + 1, // wallet1 is submitting the batch transaction
                Fee: "0",
                SigningPubKey: ""
            }
        },
        {
            RawTransaction: {
                TransactionType: "Payment",
                Flags: GlobalFlags.tfInnerBatchTxn,
                Account: wallet2.address,
                Destination: wallet3.address,
                Amount: "1000",
                Sequence: sequence2,
                Fee: "0",
                SigningPubKey: ""
            }
        }
    ],
}
```

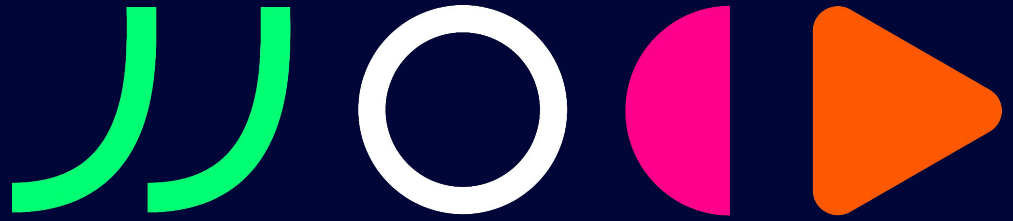
3. Prepare the batch transaction

```
const batchTxForWallet1 = structuredClone(batchTx);
const batchTxForWallet2 = structuredClone(batchTx);

signMultiBatch(wallet1, batchTxForWallet1, {batchAccount: wallet1.address});
console.log("✅ Batch transaction with wallet1 BatchSigners:", JSON.stringify(batchTxForWallet1.BatchSigners, null, 2));

signMultiBatch(wallet2, batchTxForWallet2, {batchAccount: wallet2.address});
console.log("✅ Batch transaction with wallet2 BatchSigners:", JSON.stringify(batchTxForWallet2.BatchSigners, null, 2));

const combinedTxBlob = combineBatchSigners([batchTxForWallet1, batchTxForWallet2]);
console.log("✅ Combined batch transaction blob created");

const batchTxResult = await client.submitAndWait(combinedTxBlob, { autofill: true, wallet: wallet1 });

console.log(`Transaction submitted with hash: ${batchTxResult.result.hash}`);
```

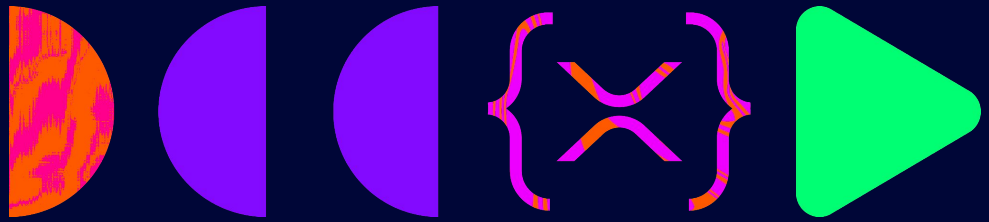
Assess your understanding 🧠

Wrap-up!

Commons



Trustlines



Tokens on XRPL: The IOU Model

A promise between parties:

- An Issuer creates a token representing an asset (like USD, EUR, or your favorite meme)
- Users establish Trustlines with issuers they want to receive tokens from
- When you hold a token, you're actually holding a promise from the issuer, the same concept behind some stablecoins (RLUSD, USDC and so)

Key Components

- **Issuer:** The entity backing the token's value and responsible for redemption
- **Trustline:** A connection showing you trust an issuer up to a specific amount
- **Token:** The digital IOU that can be transferred between users who trust the same issuer

Real-World Example:

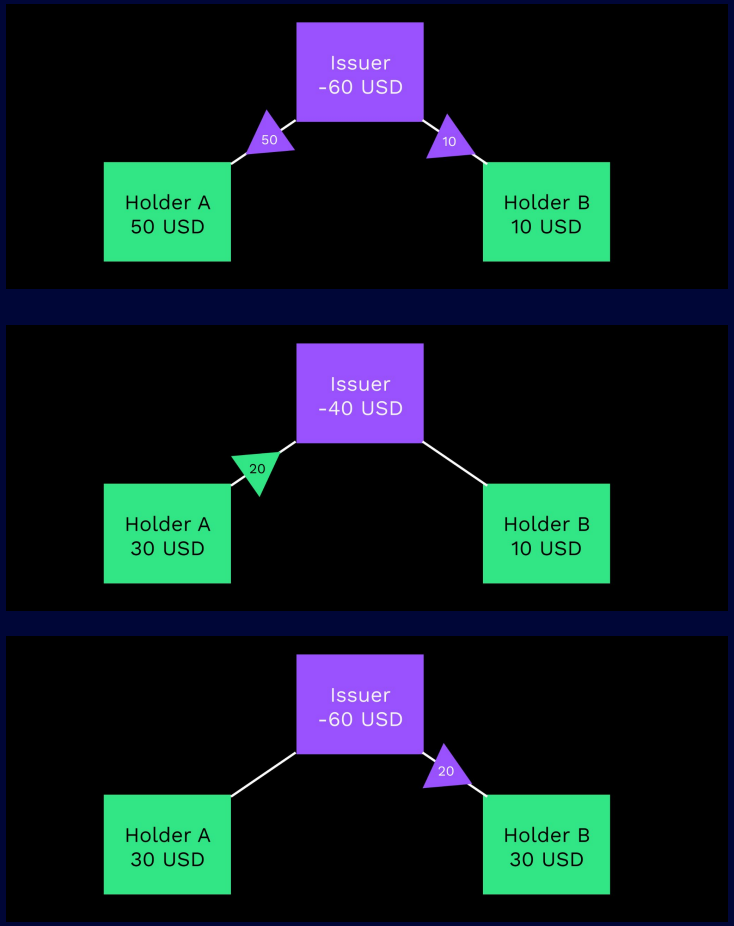
When you hold **RLUSD** on XRPL or Ethereum, you're holding **Ripple's promise to redeem 1 USD for each token**.

You can only receive these tokens after establishing **trust with Ripple as an issuer.**

Rippling

Rippling is the indirect movement of funds that occurs for any payment of fungible tokens (except when an issuing account exchanges tokens directly with another account).

This includes when one holder sends tokens to another holder of the same token.



Interacting with Trustlines

Set up Trust
- Tell the network that you trust an issuer

Get Tokens
- Buy from the Dex or the AMM
- Get some from the issuer

Use your Tokens
- Send to others
- Remove trust when done
- Provide liquidity

Source: Ripple Open Source Projects, https://hooks.xrpl.org/

1. Enable Rippling on the Issuer Account

```
async function enableRippling({ wallet, client }: any) {
  const accountSet: AccountSet = {
    TransactionType: 'AccountSet',
    Account: wallet.address,
    SetFlag: AccountSetAsfFlags.asfDefaultRipple
  }

  const prepared = await client.autofill(accountSet)
  const signed = wallet.sign(prepared)
  const result = await client.submitAndWait(signed.tx_blob)

  console.log(result)
  console.log('Enable rippling tx: ', result.result.hash)

  return
}
```

2. Receiver Account set the Trustline

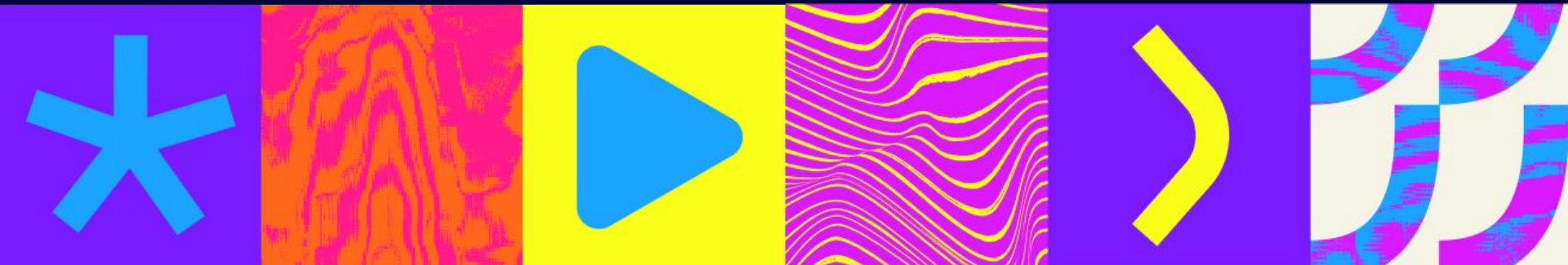
```
const trustSet: TrustSet = {
  TransactionType: 'TrustSet',
  Account: receiver.address,
  LimitAmount: {
    currency: tokenCode,
    issuer: issuer.address,
    value: '500000000' // 500M tokens
  },
  Flags: TrustSetFlags.tfClearNoRipple
}
console.log(trustSet)

// Receiver opening trust lines
const preparedTrust = await client.autofill(trustSet)
const signedTrust = receiver.sign(preparedTrust)
const resultTrust = await client.submitAndWait(signedTrust.tx_blob)

console.log(resultTrust)
console.log('Trust line issuance tx result: ', resultTrust.result.hash)
```

3. Send Initial Payment to Receiver Account

```
// Send the token to the receiver
const sendPayment: Payment = {
  TransactionType: "Payment",
  Account: issuer.address,
  Destination: receiver.address,
  Amount: {
    currency: tokenCode,
    issuer: issuer.address,
    value: "200000000", // 200M tokens
  },
};
console.log(sendPayment);

const preparedPayment = await client.autofill(sendPayment);
const signedPayment = issuer.sign(preparedPayment);
const resultPayment = await client.submitAndWait(signedPayment.tx_blob);

console.log(resultPayment);
console.log("Transfer issuance tx result: ", resultPayment.result.hash);
```

Create a token

Here is a code example.



1. Enable Rippling on the Issuer Account

```
async function enableRippling({ wallet, client }: any) {
  const accountSet: AccountSet = {
    TransactionType: 'AccountSet',
    Account: wallet.address,
    SetFlag: AccountSetAsfFlags.asfDefaultRipple
  }

  const prepared = await client.autofill(accountSet)
  const signed = wallet.sign(prepared)
  const result = await client.submitAndWait(signed.tx_blob)

  console.log(result)
  console.log('Enable rippling tx: ', result.result.hash)

  return
}
```

2. Receiver Account set the Trustline

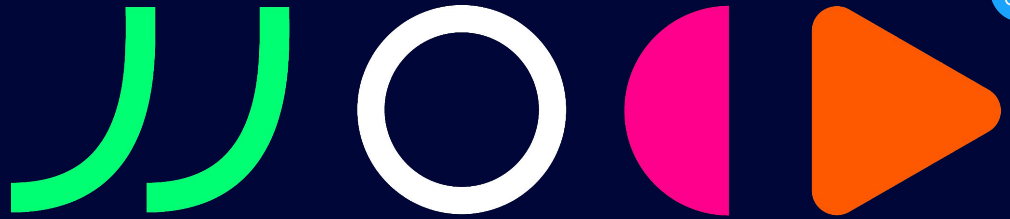
```
const trustSet: TrustSet = {
  TransactionType: 'TrustSet',
  Account: receiver.address,
  LimitAmount: {
    currency: tokenCode,
    issuer: issuer.address,
    value: '500000000' // 500M tokens
  },
  Flags: TrustSetFlags.tfClearNoRipple
}
console.log(trustSet)

// Receiver opening trust lines
const preparedTrust = await client.autofill(trustSet)
const signedTrust = receiver.sign(preparedTrust)
const resultTrust = await client.submitAndWait(signedTrust.tx_blob)

console.log(resultTrust)
console.log('Trust line issuance tx result: ', resultTrust.result.hash)
```

3. Send Initial Payment to Receiver Account

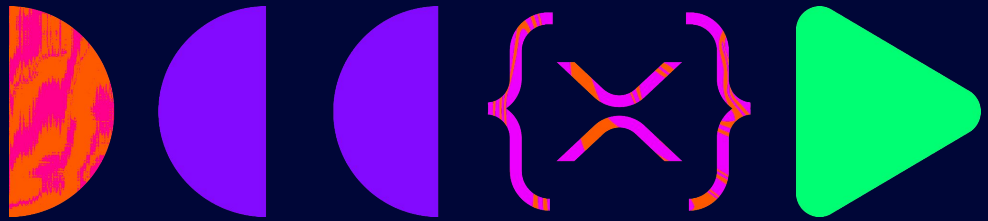
```
// Send the token to the receiver
const sendPayment: Payment = {
  TransactionType: "Payment",
  Account: issuer.address,
  Destination: receiver.address,
  Amount: {
    currency: tokenCode,
    issuer: issuer.address,
    value: "200000000", // 200M tokens
  },
};
console.log(sendPayment);

const preparedPayment = await client.autofill(sendPayment);
const signedPayment = issuer.sign(preparedPayment);
const resultPayment = await client.submitAndWait(signedPayment.tx_blob);

console.log(resultPayment);
console.log("Transfer issuance tx result: ", resultPayment.result.hash);
```

Assess your understanding 🧠

Wrap-up!



Decentralized Exchanges (DEX) 💱

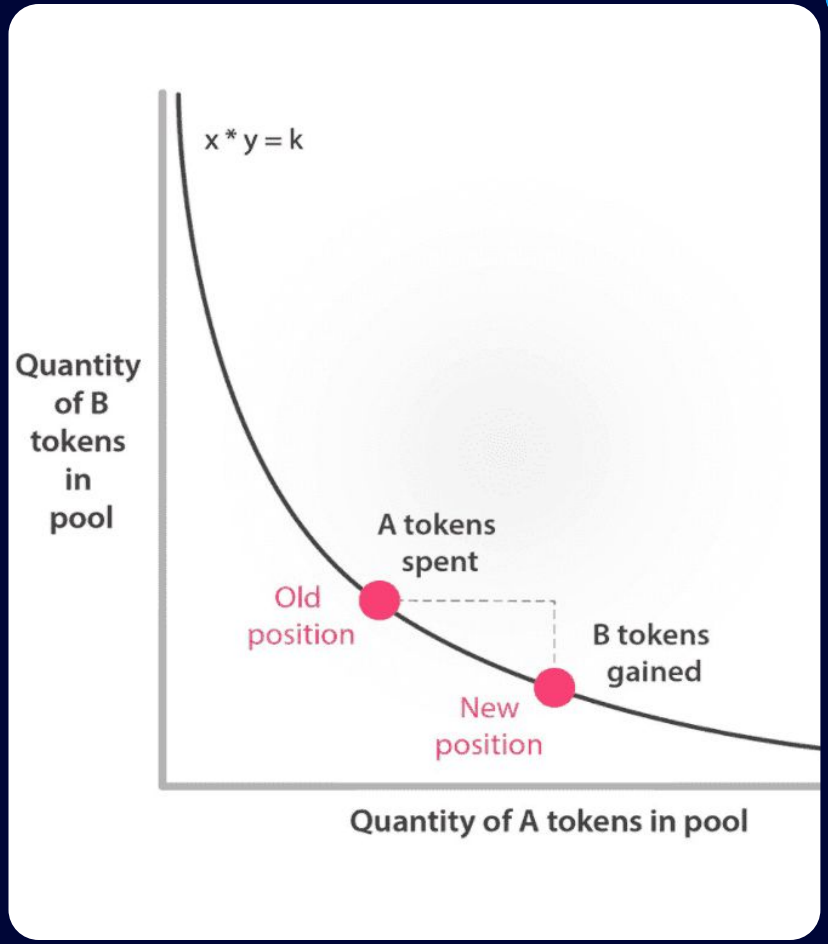


Central Limit OrderBook

- Here since day 1
- A matching system for buy and sell orders at specific prices
- Orders are stored on-ledger until filled or canceled
- Acts like a traditional exchange with bid/ask spreads

Automated Market Maker

- Added through xls-30d
- Price determined by constant product formula ($x \times y = k$)
- No counterparty needed - trade against the pool
- Liquidity providers earn fees from all trades

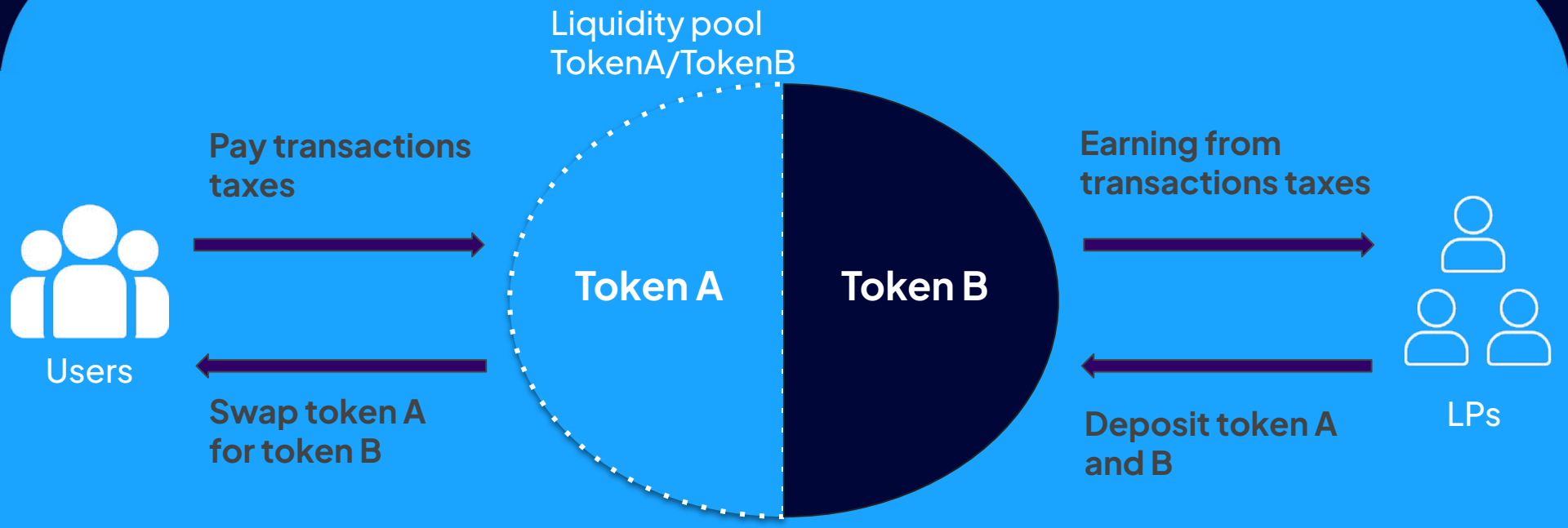


Why Use an AMM ?

- Liquidity
  - Simple mechanism for users to trade assets
  - Allow for constant liquidity and availability on the pools
  - Ideal for less liquid markets
  - All orders are market orders
  - Slippage (depends on pool size)

- Farming
  - Create add initial liquidity to the pool
  - LP represents the proportion ownership of the pool
  - Provide liquidity to AMM
  - Anyone can be market maker
  - Passively generate returns on liquidity

AMM Flow


Liquidity pool
TokenA/TokenB
Pay transactions taxes
Earning from transactions taxes
Token A
Token B
Users
LPs
Swap token A for token B
Deposit token A and B

Constant Product Market Maker

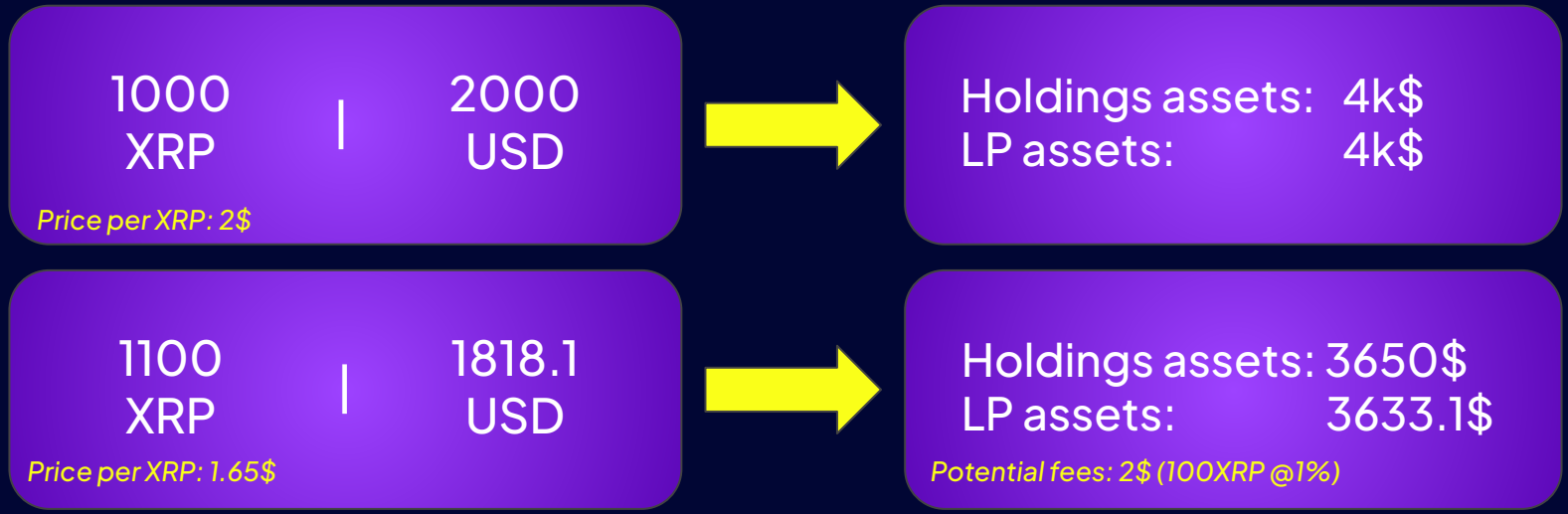
X * Y = K

1000 XRP x	x	2000 USD y	=	2 000 000 k	Price per XRP 2 USD
+100 XRP	SWAP	-181.9 USD			
1100 XRP x	x	1818.1 USD y	=	2 000 000 k	Price per XRP 1,65 USD

Commons

Impermanent loss (IL)

Impermanent loss occurs when the price of assets in liquidity pool changes after depositing them, leading to potential losses compared to holding the assets outside the pool. It must be mitigated with AMM pool fees.



Uniques features of XRPL AMMs

Protocol Native

Developers can utilize native AAM functionality without the need to manage their own smart contract design

Continuous Auction Mechanism

Reduces impermanent loss by incentivizing arbitrageurs to bid for price discrepancies at near zero fees

CLOB integration

Price optimization determines best price through liquidity pool or central limit order book

Single-sided liquidity

Anyone can easily participate as a liquidity provider as only a single asset is required to contribute to a pool

Prevent gas front running

XRPL's deterministic random transaction ordering makes front running difficult. No miners gas prioritization.

XRPL's AMM Basic functions

AMMCreate


```
{
    "Account" : "rJVUeRqDFNs2xqA7ncVE6ZoAhPUoaJJSQm",
    "Amount" : {
        "currency" : "TST",
        "issuer" : "rP9jPyP5kyvFRb6ZiRghAGw5u8SGAmU4bd",
        "value" : "25"
    },
    "Amount2" : "250000000",
    "Fee" : "2000000",
    "Flags" : 2147483648,
    "Sequence" : 6,
    "TradingFee" : 500,
    "TransactionType" : "AMMCreate"
}
```


→ Initiate liquidity pool on the ledger

→ Create initial balances of LP for sender

→ One or both can be tokens

XRPL's AMM Basic functions

AMMDeposit


```
{
    "Account" : "rJVUeRqDFNs2xqA7ncVE6ZoAhPUoaJJSQm",
    "Amount" : {
        "currency" : "TST",
        "issuer" : "rP9jPyP5kyvFRb6ZiRghAGw5u8SGAmU4bd",
        "value" : "2.5"
    },
    "Amount2" : "30000000",
    "Asset" : {
        "currency" : "TST",
        "issuer" : "rP9jPyP5kyvFRb6ZiRghAGw5u8SGAmU4bd"
    },
    "Asset2" : {
        "currency" : "XRP"
    },
    "Fee" : "10",
    "Flags" : 1048576,
    "Sequence" : 7,
    "TransactionType" : "AMMDeposit"
}
```


→ Add liquidity to existing pool in exchange of LP tokens

→ Can be single or double assets deposits

→ Use the pool to trade for single asset deposits

XRPL's AMM Basic functions

AMMWithdraw


```
{
    "Account" : "rJVUeRqDFNs2xqA7ncVE6ZoAhPUoaJJSQm",
    "Amount" : {
        "currency" : "TST",
        "issuer" : "rP9jPyP5kyvFRb6ZiRghAGw5u8SGAmU4bd",
        "value" : "5"
    },
    "Amount2" : "50000000",
    "Asset" : {
        "currency" : "TST",
        "issuer" : "rP9jPyP5kyvFRb6ZiRghAGw5u8SGAmU4bd"
    },
    "Asset2" : {
        "currency" : "XRP"
    },
    "Fee" : "10",
    "Flags" : 1048576,
    "Sequence" : 10,
    "TransactionType" : "AMMWithdraw"
}
```


→ Withdraw assets by returning LP tokens to the pool

→ Delete the AMM if both pools are emptied

→ Can be simple or double assets withdrawals

XRPL Basic functions

AMMDelete

```
{
    "Account" : "rJVUeRqDFNs2xqA7ncVE6ZoAhPUoaJJSQm",
    "Asset" : {
        "currency" : "XRP"
    },
    "Asset2" : {
        "currency" : "TST",
        "issuer" : "rP9jPyP5kyvFRb6ZiRghAGw5u8SGAmU4bd"
    },
    "Fee" : "10",
    "Flags" : 0,
    "Sequence" : 9,
    "TransactionType" : "AMMDelete"
}
```

→ Happens if there is too many trustlines to remove at once on the AMMWithdraw

→ Can delete up to 512 trustlines per call

➡️ Let's put it in a nutshell

CLOB

- More efficient capital management (sell or buy)
- Need manual adjustment of liquidity when market making
- No impermanent loss risk

AMM

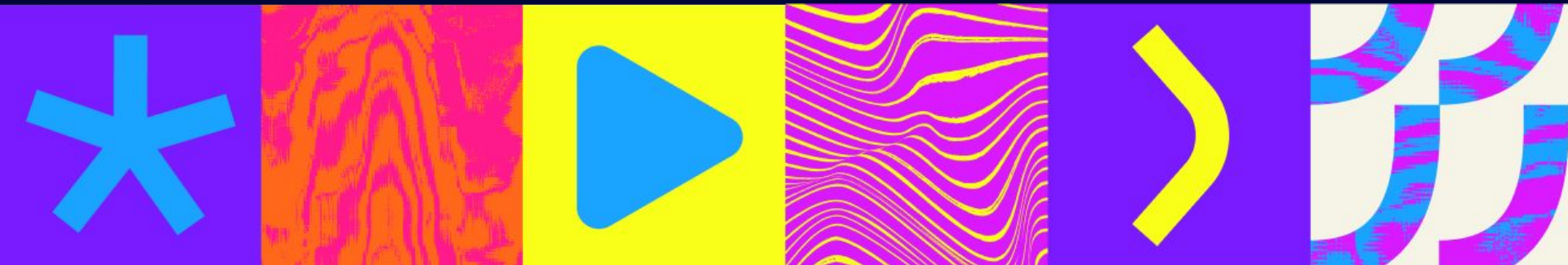
- Easiest way to exchange tokens
- Earn fees while providing liquidity
- Impermanent loss risk

PathFinding

- Allows you to get the best rates by finding the optimal route between the CLOB and the AMM

Create a Pool

Here is a code example.



Create a Pool

```
async function createAMM(
  issuer: Wallet,
  receiver: Wallet,
  client: Client,
  tokenCode: string,
) {
  console.log("create AMM", { issuer, receiver, tokenCode });
  let createAmm: AMMCreate = {
    TransactionType: "AMMCreate",
    Account: receiver.address,
    TradingFee: 600,
    Amount: {
      currency: tokenCode,
      issuer: issuer.classicAddress,
      value: "2000000", // 2M tokens
    },
    Amount2: "50000000", // 50 XRP in drops
  };
  console.log(createAmm);

  const prepared = await client.autofill(createAmm);
  const signed = receiver.sign(prepared);
  const result = await client.submitAndWait(signed.tx_blob);

  console.log(result);
  console.log("Create amm tx: ", result.result.hash);

  return;
}
```

Assess your understanding 🧠

Wrap-up!

Thank you!

Commons

BREAK

PLEASE BE BACK AT 13:30

Commons

Let us know about your experience 💬

xrpl.at/tum-xrpl-day

Thank you!

Commons