Introduction to XRPL

HAKS Hackathon 2025

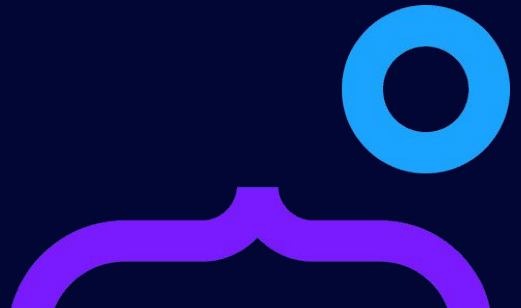
May 17th 2025

XRPL Commons

Commons | HAKS 2025

Agenda

1. Introduction to the XRP Ledger
2. Transaction Types and Flags
3. Interaction with XRPL



Introduction to the XRP Ledger

A long time ago in a galaxy far, far away....

Topic: Bitcoin without mining (Read 13555 times)

Bitcoin without mining
May 27, 2011, 03:44:53 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.

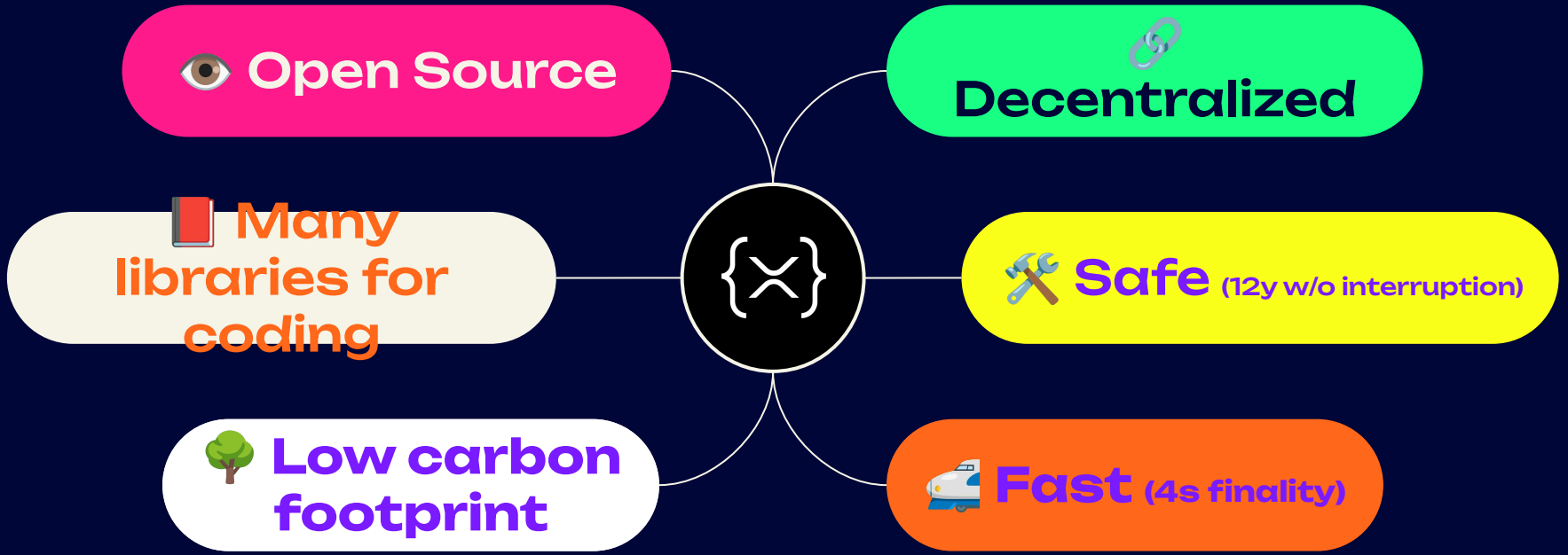
So there should always be a consensus among the trusted members of the network.

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

XRP Ledger (XRPL) launched in 2012 to address limitations of crypto and fiat currencies for financial use cases, specifically payments

The XRP Ledger



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

125+

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

2.8B+

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

~$180B+

market capitalization of XRP, making it the ~8th largest cryptocurrency

Source: CoinMarketCap, April 2024 https://livenet.xrpl.org/

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.

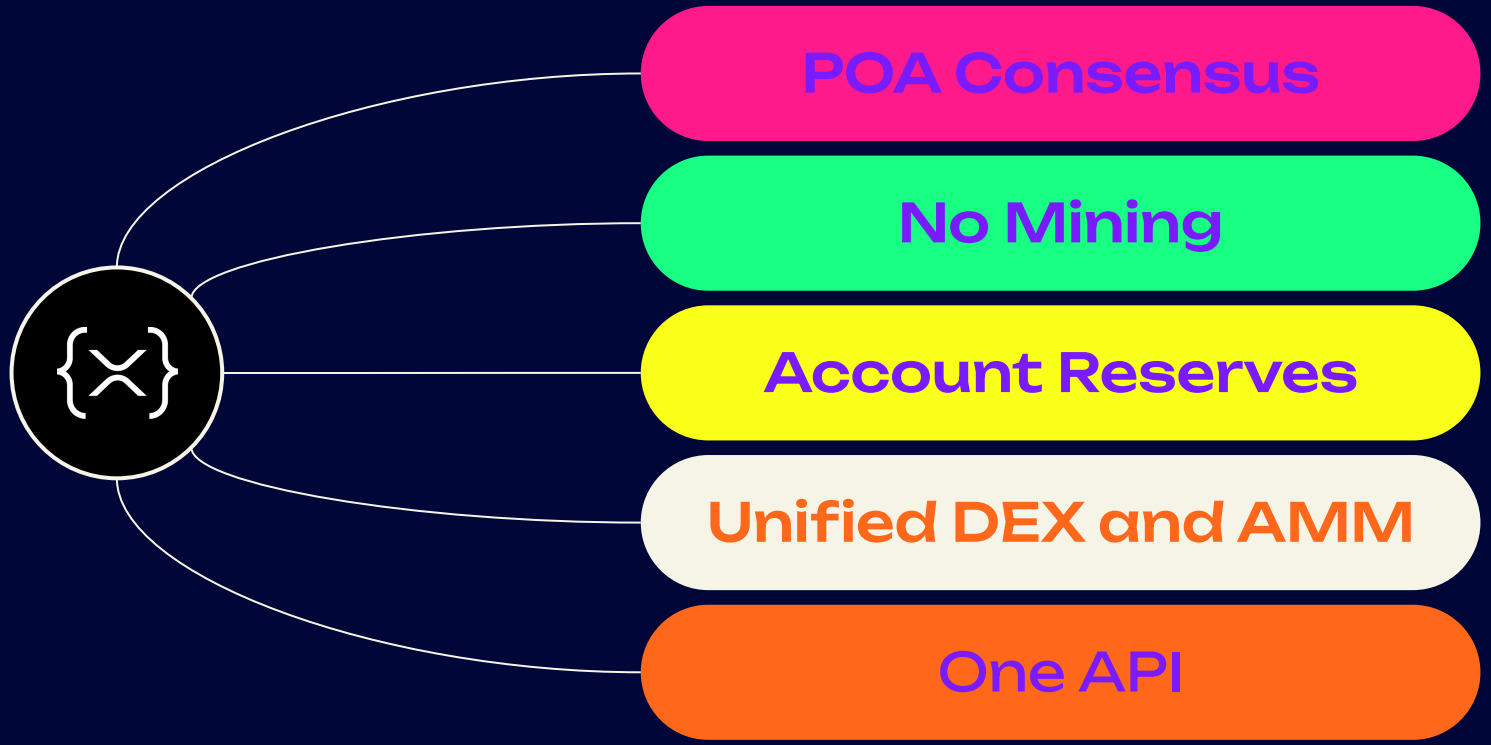
ripple

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.

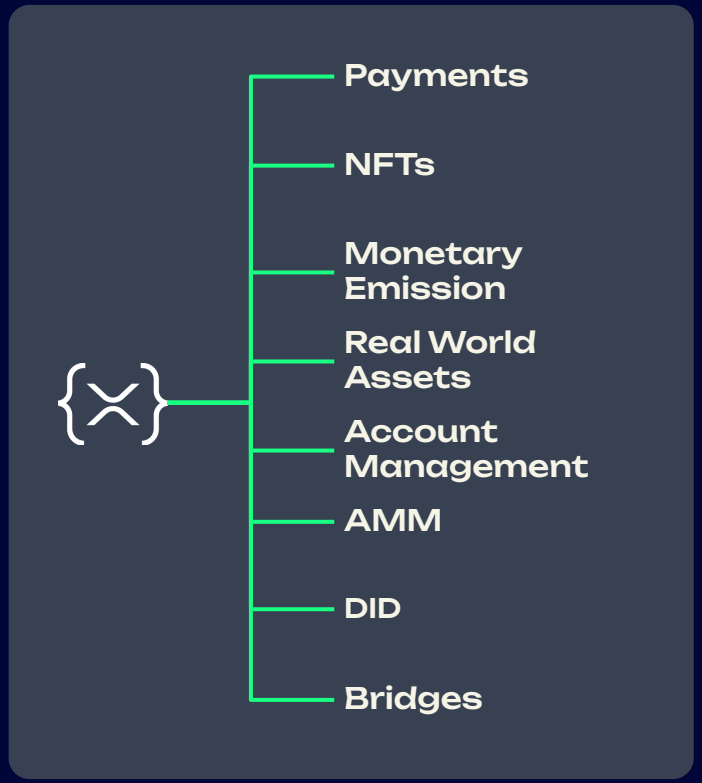
Ripple is one of many companies building on and contributing to the XRP Ledger.

Specificities of the XRPL



Transaction Types & Flags

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)

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:

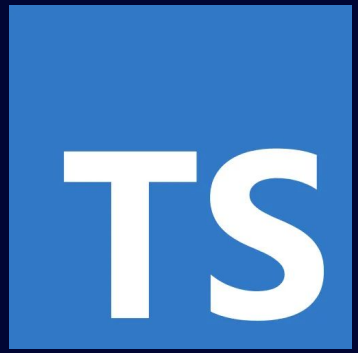
- **tfTransferable**: When disabled (0), makes NFT non-transferable
- **tfBurnable**: When enabled (1), allows NFT to be burned by issuer

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

Interaction with the XRPL

Languages with an XRPL SDK

Recommended: JavaScript/TypeScript or Python (most up to date)



Create a transaction

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();
}
```

Trustline

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

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)
```

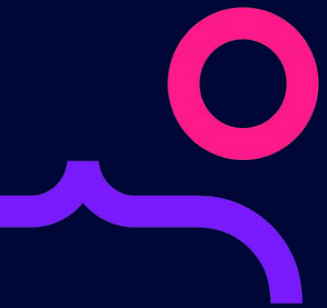
3. Send initial Payment to receiver account

```
// 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);
```

Escrow



Escrow Payment

- Contractual Agreement: Escrow is a contract between two parties for secure transactions.
- Third-Party Involvement: A neutral third party holds funds until conditions are met.
- Conditional Release: Funds are released only when specified conditions are fulfilled.

Different types of Escrow on the XRPL

Time Based

- Funds only become available after a certain amount of time passes.
- A transaction needs to be made to release the funds

Condition Based

- Escrow is created with a **condition** and **fulfillment key.**
- Condition locks funds until the correct **fulfillment key is provided**.

Combination Escrow

- Combines **time-based** and **conditional** features.
- Inaccessible **until specified time passes**, then funds can be released with the **correct key**.

Escrow Lifecycle

Create the Escrow

- Using an EscrowCreate transaction
- Define the type of escrow, the amount locked and the conditions

The Escrow is created on-chain

- Holds the XRP in the escrow account

Finish the Escrow

- The recipient send a EscrowFinish transaction
- Deliver the XRP if the conditions are meet
- Destroy the Escrow account on the ledger

Current limitation of Escrow

- Escrow **only works with XRP**, not tokens.
- While the escrow is incomplete, the sender is responsible for the reserve requirement of the Escrow object.
- You can't create an escrow with past time values.
- Timed releases and expirations resolve according to ledger close times. In practice, actual release and expiration times can vary by about five seconds as ledgers close.
- The only supported **crypto-condition type is PREIMAGE-SHA-256**.

Creating a Time Based Escrow

```
async function createTimeEscrow() {
  const client = new Client('wss://s.altnet.rippletest.net:51233');
  await client.connect();

  const finishAfter = new Date(Date.now() + WAITING_TIME);

  const escrowCreateTx: EscrowCreate = {
    Account: walletOne.address,
    TransactionType: 'EscrowCreate',
    Amount: xrpToDrops('1'),
    Destination: walletTwo.address,
    FinishAfter: isoTimeToRippleTime(finishAfter.toString()),
  };

  const tx = await client.submitAndWait(
    walletOne.sign(await client.autofill(escrowCreateTx)).tx_blob
  );
  await client.disconnect();
  return tx.result.tx_json.Sequence;
}
```

Finishing a Time Based Escrow

```
async function finishEscrow(sequence: number) {
  const client = new Client('wss://s.altnet.rippletest.net:51233');
  await client.connect();

  const escrowFinishTx: EscrowFinish = {
    Account: walletTwo.address,
    TransactionType: 'EscrowFinish',
    OfferSequence: sequence,
    Owner: walletOne.address,
  };
  console.log(escrowFinishTx);

  const tx = await client.submitAndWait(
    walletTwo.sign(await client.autofill(escrowFinishTx)).tx_blob
  );
  console.log(tx);
}
```

Generating the Fulfillment

```
function generateConditionAndFulfillment() {
  const preimage = crypto.randomBytes(32);

  const fulfillment = new cc.PreimageSha256();
  fulfillment.setPreimage(preimage);

  const condition = fulfillment
    .getConditionBinary()
    .toString('hex')
    .toUpperCase();

  console.log('Condition:', condition);

  const fulfillment_hex = fulfillment
    .serializeBinary()
    .toString('hex')
    .toUpperCase();

  console.log(
    'Fulfillment (keep secret until you want to finish the escrow):',
    fulfillment_hex
  );

  return {
    condition,
    fulfillment: fulfillment_hex,
  };
}
```

Creating the condition based Escrow

```
async function createConditionEscrow(condition: string) {
  const client = new Client('wss://s.altnet.rippletest.net:51233');
  await client.connect();
  const finishAfter = new Date(Date.now() + 100000000);
  console.log(isoTimeToRippleTime(finishAfter.toString()));

  const escrowCreateTx: EscrowCreate = {
    Account: walletTwo.address,
    TransactionType: 'EscrowCreate',
    Amount: '1',
    Destination: walletOne.address,
    Condition: condition,
    CancelAfter: isoTimeToRippleTime(finishAfter.toString()),
  };

  const tx = await client.submitAndWait(
    walletTwo.sign(await client.autofill(escrowCreateTx)).tx_blob
  );
  console.log(tx);
  await client.disconnect();
  return tx.result.tx_json.Sequence;
}
```

Finishing a condition based Escrow

```
async function finishEscrow(sequence: number, fulfillment: string) {
  const client = new Client('wss://s.altnet.rippletest.net:51233');
  await client.connect();

  const escrowFinishTx: EscrowFinish = {
    Account: walletOne.address,
    TransactionType: 'EscrowFinish',
    OfferSequence: sequence,
    Owner: walletTwo.address,
    Fulfillment: fulfillment,
  };

  const tx = await client.submitAndWait(
    walletOne.sign(await client.autofill(escrowFinishTx)).tx_blob
  );
  console.log(tx);
  return;
}
```

Thank you for listening!

Resources

xrpl.at/haks2025