OpenPeer

OpenPeer Litepaper V0.3

March 2023

Josh Reyes & Marcos Teixeira

Abstract

OpenPeer is a decentralized peer-to-peer (P2P) protocol for people to exchange crypto assets like Ethereum and USDT (Tether) for fiat money without centralized counterparty risk through escrow smart contracts. The protocol allows people from any country to exchange assets on any smart contract-supporting blockchain network and with any fiat money payment network or bank transfer method.

The protocol is governed by a decentralized autonomous organization (DAO) that also arbitrates disputes between parties. Stakeholders coordinate through the protocol token, but OpenPeer users will not need to interact with the protocol token in order to complete an exchange. The dispute mechanism relies on game theory incentives for arbitrators to rule cases truthfully and to prevent bad actors.

Introduction

Crypto adoption has been accelerating throughout the last decade, especially in developing countries experiencing sustained inflation and depreciation of their fiat currencies against the US dollar. This has been seen in the explosive growth of US dollar-pegged stablecoins which have increased 1000% in market capitalization over the past five years[1]. However one of the largest challenges remaining in crypto asset adoption in emerging markets has been onboarding people in developing countries onto blockchain networks. Traditional onramp methods through card networks or bank transfer methods often don't exist or are plagued by high fees and failure rates that severely hinders crypto adoption.

This has led to the majority of crypto users in emerging markets exchanging crypto assets and fiat money through peer-to-peer (P2P) platforms. These platforms are largely offered by centralized exchanges which have always been the subject of heists and fraud leading to the loss of customer funds. In order for more users to adopt self-custody wallets in the following years, a decentralized P2P protocol needs to be available to onboard users into crypto assets.

History of P2P Platforms

Since the emergence of cryptocurrency, some early adopters have been looking for alternative options to centralized exchange trading. Centralized exchanges, which keep records of customer identities and balances in a database and facilitate the transfer of fiat money to and from cryptocurrency, are seen by some as contradictory to the decentralized nature of cryptocurrencies.

One alternative to centralized exchange trading is over-the-counter (OTC) trading. In OTC trading, two parties exchange directly with one another without the oversight of an exchange. This type of trading offers flexibility as there are no limitations on what can be exchanged for cryptocurrency and the transactions are peer-to-peer, without the need for a financial intermediary. Additionally, fees for larger transactions can be lower than exchange fees.

However, OTC trading also carries a significant risk, as both parties may be uncertain about the other's ability to fulfill their end of the trade and not abscond with the money or cryptocurrency.

[1] Coingecko - Top Stablecoins by Marketcap. https://www.coingecko.com/en/categories/stablecoins

An early innovation in OTC trading was the creation of an online "web of trust" platform, like the one known as #bitcoin-otc. On this platform, traders can leave public ratings for other traders, and a website will calculate each user's cumulative trust score. If a user has a high rating, they are more reputable and less likely to scam. While this system was useful, it was not perfect as it had some loopholes, like high-rated traders committing heists and scammers manipulating their ratings.

In 2012, LocalBitcoins introduced an escrow system as a solution. Sellers of Bitcoin could now transfer their BTC to LocalBitcoins first and release it from escrow once the buyer confirms payment. If the buyer fails to send the money, the seller can request LocalBitcoins to return the coins. On the other hand, if the seller refuses to release the escrow, the buyer can request LocalBitcoins to review their proof of payment and force the seller to release the Bitcoins. This system combined with the reputation system worked well but the problem was that LocalBitcoins became the new intermediary. If LocalBitcoins were hacked, went offline or its founders decided to abscond, all the Bitcoins in escrow could be lost. There was a huge amount of trust placed in the centralized escrow system and users no longer held the keys to their Bitcoin; instead, they were trusting LocalBitcoins to hold Bitcoins for them, similar to a bank or an exchange. This moved the counter-party risk but it was not completely eliminated.

The launch of Ethereum made it possible to program money and as a result, many ideas not possible with Bitcoin could now be achieved through Ethereum smart contracts. Building on the concepts of a web of trust and escrow system, OpenPeer is developing a P2P OTC trading platform that keeps users in control of their cryptocurrency at all times.

Disadvantages of Existing Onramp Solutions

In order to understand the need for OpenPeer, a look at the current state of crypto on/off ramps is necessary. For self-custody wallets the most popular onramp providers are MoonPay, Ramp Network and Wyre. Though existing onramps have been a popular addition to self-custody wallets over the past years, their success is largely limited to developed countries for the following reasons.

Limited Availability of Payment Methods

Unlike developed countries, alternative payment networks are largely the norm in developing countries. For example the instant-transfer networks PIX and UPI in Brazil and India, are both the dominant payment methods for consumers in each respective country. However few onramp providers offer these payment methods, and when they do the fees are exorbitantly high, often near 3% of the total transactions size[2]. Because OpenPeer connects two parties, any payment method to transfer fiat currency can be used and the fees are a tenth of existing onramp solutions.

Payment Destinations are a Central Point of Failure

Even when alternative payment methods are available, payments to the provider entities from a crypto buyer's bank are often blocked. As local currencies devalue, it is not in the bank’s self interest to have

[2] Moonpay Help Center - What fees do you charge?
https://support.moonpay.com/hc/en-gb/articles/360011930117-What-fees-do-you-charge-

their depositors move funds to crypto. Only by decentralizing the exchanges between crypto assets and fiat money through a P2P platform are people able to reliably onboard into the crypto economy.

Limited Availability of Countries

Some of the largest crypto adopting countries are not served by leading onramp providers today. For example Pakistan with 220 million people and ranked 6th on Chainalysis' Global Crypto Index[3] isn't available with market leaders like Moonpay, Wyre and Ramp Network[4]. Regulatory uncertainty and the need for centralized entities to find banking partners in each country also makes it difficult for centralized on/off ramp providers to provide services in developing countries where crypto is popular. Because OpenPeer doesn't require banking partners, any country can be supported globally by the protocol. However the front-end served by any legal entities may require geo-blocking if required in the jurisdictions in which we are domiciled.

High Fees

Most on/off ramps today charge transaction fees of 3% on average. There are also minimum fees per transaction on average between $4-5. In developing countries where the average transaction can be $50 for new users, this results in a major barrier to entry as users are required to pay an effective transaction fee of 10% along with any additional taxes and fees required by their government or bank. The high transaction fee removes the economic incentive for buyers to switch their fiat money for crypto assets. OpenPeer doesn't charge any fees to crypto buyers and only takes a 0.3% fee from sellers.

The Disadvantages of P2P Platforms

The leading P2P platforms today are offered by centralized exchanges including but not limited to Binance, Kucoin and OKX.

Counterparty Risk

The collapse of FTX in late 2022 showed the risks with centralized exchanges. As a P2P merchant selling crypto assets, you need to keep all your available funds on the centralized exchange in order to exchange with other users. In prior years, centralized exchanges have been the victim of hacks leading to the loss of user funds. Phishing attacks are also another risk when funds are kept on a centralized exchange.

As a user you are also required to receive the funds on a centralized exchange resulting in the same risk that merchants are exposed to. Though you can later move your purchased crypto assets to a self-custody wallet, the extra step is cumbersome for inexperienced users. With OpenPeer users can directly fund their self-custody wallet without interacting with a centralized middleman.

---

[3] Chainalysis - 2022 Global Crypto Adoption Index.
https://blog.chainalysis.com/reports/2022-global-crypto-adoption-index/
[4] Moonpay - What are our non-supported countries, states and territories for on-ramp product?
https://support.moonpay.com/hc/en-gb/articles/6557330712721-What-are-our-non-supported-countries-states-and-territories-for-on-ramp-product-

Privacy

Another disadvantage of P2P platforms offered by centralized exchanges is the need to verify your identity. Centralized exchanges holding all customer identity and financial records create a honeypot for hackers looking to exploit this information. In 2018, popular Brazilian crypto investment platform Atlas Quantum was the victim of a hack that resulted in 261,000 customers' records being leaked including emails, names and physical addresses[5]. In late 2022, popular American exchange Gemini leaked 5.7 million customer emails and phone numbers[6]. For centralized exchange P2P platforms, personal details are used to provide a reputation signal to other users that a user is legitimate along with preventing sybil attacks. In OpenPeer, decentralized identity and proof of humanity solutions will be utilized in order to achieve the same results while allowing users to preserve their privacy.

How Trading on OpenPeer Works

Definitions

We need to define names for participants and actions in order to explain how people use OpenPeer.

Seller: Person posting an advertisement that they want to sell crypto and exchanging said crypto assets for fiat money

Buyer: Person responding to advertisement and purchasing crypto assets with fiat money

Trade: Exchange between merchant and buyer

An OpenPeer Trade

As OpenPeer never holds funds, the experience differs from centralized exchanges and escrow providers. OpenPeer doesn't take deposits, process withdrawals or hold fiat or crypto. Exchanging fiat money for crypto assets on OpenPeer looks like the scenarios below:

1. Alice (The Seller) posts an ad to sell USDT, posting a message to a OpenPeer contract accessible and public for anyone else to find. The ad states the available USDT to sell, minimum and maximum trade sizes, acceptable payment methods, and any price rules. For example: Alice is selling 1000 USDT, with a minimum sale of 100 USDT and maximum of 1000 USDT, she is willing to accept ₹80 per USDT.

2. Bob (The Buyer) responds to the ad to exchange USDT for INR ₹. When Bob responds to the offer, Alice is sent a notification to escrow the funds for the trade. Bob wants to purchase 100 USDT for ₹8000.

---

[5] Cointelegraph - Brazilian Crypto Platform Atlas Quantum Reveals Data Breach Affecting 260K Customers. https://cointelegraph.com/news/brazilian-crypto-platform-atlas-quantum-reveals-data-breach-affecting-260k-customers

[6] Cointelegraph - 'Third-party incident' impacted Gemini with 5.7 million emails leaked. https://cointelegraph.com/news/gemini-allegedly-suffers-data-breach-5-7-million-emails-leaked

3. Alice accepts the offer and in a single transaction deploys a contract that escrows her funds. Once funds are escrowed, Bob receives a notification revealing Alice's payment details and instructions on how to make the payment. As soon as the crypto is escrowed a payment window of 24 hours begins. If payment is not made within that time the crypto escrowed in the contract can be released back to the seller.

4. Bob confirms when he makes the payment and signs a transaction so the escrowed funds cannot be withdrawn back to the seller without arbitration. A notification is sent to Alice that payment has been made and to release the escrowed USDT to Bob.

5. Upon receipt of ₹8000, Alice signs a transaction to release the escrowed funds from the contract and Bob receives 100 USDT.

An OpenPeer Dispute

After crypto has been escrowed either party can raise a dispute and assign a third-party arbitrator to make a resolution. In order to initiate a dispute both parties must pay a small fee to initiate arbitration. At this time they can also submit evidence such as proof of payment to help the arbitrators make a decision.

At launch the arbitrator will be a multisig controlled by the OpenPeer team. As the protocol grows arbitration will be decentralized to the community and arbitration will be done by protocol token stakers.

OpenPeer Architecture

Accounts

Accounts in OpenPeer are represented by each user's Ethereum address also known as their externally-owned-account (EOA). Users can login to OpenPeer through a Metamask, Minke, or any other EVM-compatible wallet. OpenPeer will develop and maintain SDKs so any wallet can integrate with the protocol. Users are not required to share any other piece of identity, though proof-of-humanity and other decentralized identity solutions will form an important reputation layer. protocol token stakers may be able to validate proof-of-humanity and other identity solutions in the future to earn fees.

Escrow Creation

Most peer-to-peer trades in emerging markets today happen through US dollar pegged stablecoins, primarily with USDT (Tether). Though OpenPeer is a self-custody platform, it is built to compete with centralized platforms in terms of user experience. As Ethereum increases in popularity, more people are moving to Layer2 scaling solutions or alternative Layer 1 blockchains including Polygon,

Binance Smart Chain and Arbitrum. With lower gas-fees, OpenPeer uses decentralized relayers like Biconomy in order to cover gas fees for users. Therefore users can utilize the protocol with only stablecoins in their wallet without the need for the native token like Ether, MATIC or BNB.

In order to deploy an escrow contract, a seller interacts with the OpenPeer Deployer contract. The OpenPeer Deployer contract then automatically deploys the individual escrow contract holding their funds. By requiring escrow contracts to be deployed through the deployer contract, OpenPeer can identify which contracts are created through the protocol and ensure no phishing functions are added by bad actors.

Reputation

At launch reputation will be centralized and based on core metrics including:

Wallet Age: How long the wallet address has been making transactions on Ethereum, Polygon or any EVM chain

Account Age: How long the account has been created for

Trade Count: How many successful trades have been completed through OpenPeer

Volume: Historical volume traded denominated in US dollars

Positive Feedback Percentage: After every trade each user will have the option to rate their trade as Good or Bad

OpenPeer intends to decentralize reputation over time so any front-end application can interact with all aspects of the protocol. Importantly, users will have the option to decide which core metrics they choose to display to retain privacy. Reputation-based NFTs and soulbound tokens provide possible future paths to bring reputation on-chain.

Communication Between Parties

OpenPeer strives to maintain the highest levels of privacy between parties. Messaging between parties will be end-to-end encrypted. Only when a dispute is opened will a key be created in order for arbitrators to see messages. Users can opt to receive trade notifications via email or PGP. As wallet-to-wallet messaging and communication platforms improve over the coming years, OpenPeer will seek to integrate them into the protocol.

OpenPeer Contracts

EscrowDeployer

This following contract is the OpenPeerEscrowsDeployer and is responsible for deploying individual escrow contracts.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

import { OpenPeerEscrow } from "./OpenPeerEscrow.sol";
import { Ownable } from "./libs/Ownable.sol";
import { ERC2771Context } from "./libs/ERC2771Context.sol";

contract OpenPeerEscrowsDeployer is ERC2771Context, Ownable {
    mapping (address => bytes32) public escrows;

    /**********************
    +   Global settings   +
    **********************/
    address public arbitrator;
    address public feeRecipient;
    uint8 public fee;

    bool public stopped = false;

    /**********************
    +   Events            +
    **********************/
    event EscrowCreated(address indexed _escrow, bytes32 indexed _tradeHash);

    /// @notice Settings
    /// @param _arbitrator Address of the arbitrator (currently OP staff)
    /// @param _feeRecipient Address to receive the fees
    /// @param _fee OP fee (bps) ex: 30 == 0.3%
    /// @param _trustedForwarder Forwarder address
    constructor (
        address _arbitrator,
        address _feeRecipient,
        uint8 _fee,
        address _trustedForwarder
    ) ERC2771Context(_trustedForwarder) {
        arbitrator = _arbitrator;
        feeRecipient = _feeRecipient;
        fee = _fee;
    }

    /**********************
    +   Modifiers          +
    **********************/

    // circuit breaker modifiers
    modifier stopInEmergency {
        if (stopped) {
            revert("Paused");
        } else {
            _;
        }
    }

    function deployNativeEscrow(address _buyer, uint256 _amount) external
stopInEmergency {
        deploy(_buyer, address(0), _amount);
    }

    function deployERC20Escrow(address _buyer, address _token,  uint256 _amount)
external stopInEmergency {
        deploy(_buyer, _token, _amount);
    }

    function deploy(address _buyer, address _token, uint256 _amount) private {
        OpenPeerEscrow escrow = new OpenPeerEscrow(_buyer,
                                                   _token,
                                                   _amount,
                                                   fee,
```

```
                                                    arbitrator,
                                                    feeRecipient,
                                                    _trustedForwarder);

        bytes32 _tradeHash = keccak256(
            abi.encodePacked(address(escrow),
                             _msgSender(),
                             _buyer,
                             _token,
                             _amount,
                             fee,
                             arbitrator,
                             feeRecipient)
        );
        escrows[address(escrow)] = _tradeHash;
        emit EscrowCreated(address(escrow), _tradeHash);
    }

    /**********************
    +   Setters           +
    **********************/

    /// @notice Updates the arbitrator
    /// @param _arbitrator Address of the arbitrator
    function setArbitrator(address _arbitrator) public onlyOwner {
        arbitrator = _arbitrator;
    }

    /// @notice Updates the fee recipient
    /// @param _feeRecipient Address of the arbitrator
    function setFeeRecipient(address _feeRecipient) public onlyOwner {
        feeRecipient = _feeRecipient;
    }

    /// @notice Updates the fee
    /// @param _fee fee amount (bps)
    function setFee(uint8 _fee) public onlyOwner {
        fee = _fee;
    }

    /// @notice Updates the forwarder
    /// @param trustedForwarder biconomy forwarder
    function setTrustedForwarder(address trustedForwarder) external onlyOwner {
        _trustedForwarder = trustedForwarder;
    }

    /// @notice Pauses and activate the contract
    function toggleContractActive() public onlyOwner {
        stopped = !stopped;
    }

    /// @notice Version recipient
    function versionRecipient() external pure returns (string memory) {
                return "1.0";
       }
}
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

import { OpenPeerEscrow } from "./OpenPeerEscrow.sol";
import { Ownable } from "./libs/Ownable.sol";
import { ERC2771Context } from "./libs/ERC2771Context.sol";

contract OpenPeerEscrowsDeployer is ERC2771Context, Ownable {
    mapping (address => bytes32) public escrows;
```

```
    /***********************
    +   Global settings   +
    ***********************/
    address public arbitrator;
    address public feeRecipient;
    uint8 public fee;

    bool public stopped = false;

    /***********************
    +   Events            +
    ***********************/
    event EscrowCreated(address indexed _escrow, bytes32 indexed _tradeHash);

    /// @notice Settings
    /// @param _arbitrator Address of the arbitrator (currently OP staff)
    /// @param _feeRecipient Address to receive the fees
    /// @param _fee OP fee (bps) ex: 30 == 0.3%
    /// @param _trustedForwarder Forwarder address
    constructor (
        address _arbitrator,
        address _feeRecipient,
        uint8 _fee,
        address _trustedForwarder
    ) ERC2771Context(_trustedForwarder) {
        arbitrator = _arbitrator;
        feeRecipient = _feeRecipient;
        fee = _fee;
    }

    /***********************
    +   Modifiers          +
    ***********************/

    // circuit breaker modifiers
    modifier stopInEmergency {
        if (stopped) {
            revert("Paused");
        } else {
            _;
        }
    }

    function deployNativeEscrow(address _buyer, uint256 _amount) external
stopInEmergency {
        deploy(_buyer, address(0), _amount);
    }

    function deployERC20Escrow(address _buyer, address _token,  uint256 _amount)
external stopInEmergency {
        deploy(_buyer, _token, _amount);
    }

    function deploy(address _buyer, address _token, uint256 _amount) private {
        OpenPeerEscrow escrow = new OpenPeerEscrow(_buyer,
                                                   _token,
                                                   _amount,
                                                   fee,
                                                   arbitrator,
```

```
                                                feeRecipient,
                                                _trustedForwarder);

        bytes32 _tradeHash = keccak256(
            abi.encodePacked(address(escrow),
                             _msgSender(),
                             _buyer,
                             _token,
                             _amount,
                             fee,
                             arbitrator,
                             feeRecipient)
        );
        escrows[address(escrow)] = _tradeHash;
        emit EscrowCreated(address(escrow), _tradeHash);
    }

    /**********************
    +   Setters           +
    **********************/

    /// @notice Updates the arbitrator
    /// @param _arbitrator Address of the arbitrator
    function setArbitrator(address _arbitrator) public onlyOwner {
        arbitrator = _arbitrator;
    }

    /// @notice Updates the fee recipient
    /// @param _feeRecipient Address of the arbitrator
    function setFeeRecipient(address _feeRecipient) public onlyOwner {
        feeRecipient = _feeRecipient;
    }

    /// @notice Updates the fee
    /// @param _fee fee amount (bps)
    function setFee(uint8 _fee) public onlyOwner {
        fee = _fee;
    }

    /// @notice Updates the forwarder
    /// @param trustedForwarder biconomy forwarder
    function setTrustedForwarder(address trustedForwarder) external onlyOwner {
        _trustedForwarder = trustedForwarder;
    }

    /// @notice Pauses and activate the contract
    function toggleContractActive() public onlyOwner {
        stopped = !stopped;
    }

    /// @notice Version recipient
    function versionRecipient() external pure returns (string memory) {
              return "1.0";
       }
}
```

The properties of the Deployer contract are detailed below

mapping (address => bytes32) public escrows: A mapping that stores the relationship between an escrow contract address and its corresponding trade hash.

address public arbitrator: The address of the arbitrator that will be used by the escrow contracts that are deployed by this contract.

address public feeRecipient: The address that will receive the fees associated with the escrow contracts that are deployed by this contract.

uint8 public fee: The fee percentage that will be used by the escrow contracts that are deployed by this contract.

bool public stopped: A boolean variable that indicates whether or not the contract has been paused.

It has two public functions that can be called by anyone:

1. deployNativeEscrow(address _buyer, uint256 _amount): This function is used to deploy an instance of the OpenPeerEscrow contract for a native token (Native token) transfer.
2. deployERC20Escrow(address _buyer, address _token, uint256 _amount): This function is used to deploy an instance of the OpenPeerEscrow contract for an ERC20 token transfer.

It also has several functions that can be only called by contract owner:

1. setArbitrator(address _arbitrator): updates the arbitrator address.
2. setFeeRecipient(address _feeRecipient): updates the fee recipient address.
3. setFee(uint8 _fee): updates the fee percentage.
4. setTrustedForwarder(address _trustedForwarder): updates the Forwarder address
5. toggleContractActive(): pauses or activates the contract.

The deployNativeEscrow and deployERC20Escrow functions both call the private deploy function, passing in the buyer address, token address, amount, and other relevant information, like the fee percentage and the addresses of the arbitrator and fee recipient. The deploy function then creates an instance of the OpenPeerEscrow contract and creates a trade hash for the escrow. The trade hash and the escrow address are then stored in the escrow's mapping, and an EscrowCreated event is emitted.

Example Escrow Contract

The following is an example of an Escrow Contract deployed by the EscrowDeployer

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

import { IERC20 } from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import { SafeERC20 } from "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import { ERC2771Context } from "./libs/ERC2771Context.sol";

contract OpenPeerEscrow is ERC2771Context {
    using SafeERC20 for IERC20;
    // Settings
    // Address of the arbitrator (currently OP staff)
    address public arbitrator;
```

```
    // Address to receive the fees
    address public feeRecipient;
    address public immutable seller;
    address public immutable buyer;
    address public immutable token;
    uint256 public immutable amount;
    uint32 public immutable fee;
    uint32 public sellerCanCancelAfter;

    bool public dispute;

    /// @notice Settings
    /// @param _buyer Buyer address
    /// @param _token Token address or 0x0000000000000000000000000000000000000000 for
native token
    /// @param _fee OP fee (bps) ex: 30 == 0.3%
    /// @param _arbitrator Address of the arbitrator (currently OP staff)
    /// @param _feeRecipient Address to receive the fees
    /// @param _trustedForwarder Forwarder address
    constructor(
        address _buyer,
        address _token,
        uint256 _amount,
        uint8 _fee,
        address _arbitrator,
        address _feeRecipient,
        address _trustedForwarder
    ) ERC2771Context(_trustedForwarder) {
        require(_amount > 0, "Invalid amount");
        require(_buyer != _msgSender(), "Seller and buyer must be different");
        require(_buyer != address(0), "Invalid buyer");

        seller = _msgSender();
        token = _token;
        buyer = _buyer;
        amount = _amount;
        fee = uint32(amount * _fee / 10_000);
        arbitrator = _arbitrator;
        feeRecipient = _feeRecipient;
    }

    // Events
    event Created();
    event Released();
    event CancelledByBuyer();
    event SellerCancelDisabled();
    event CancelledBySeller();
    event DisputeOpened();
    event DisputeResolved();

    modifier onlySeller() {
        require(_msgSender() == seller, "Must be seller");
        _;
    }

    modifier onlyArbitrator() {
        require(_msgSender() == arbitrator, "Must be arbitrator");
        _;
    }

    modifier onlyBuyer() {
        require(_msgSender() == buyer, "Must be buyer");
        _;
    }

    /// @notice Create and fund a new escrow.
    function escrow() payable external {
```

```
    require(sellerCanCancelAfter == 0, "Funds already escrowed");
    if (token == address(0)) {
        require(msg.value == amount + fee, "Incorrect MATIC sent");
    } else {
        IERC20(token).safeTransferFrom(_msgSender(), address(this), amount + fee );
    }

    sellerCanCancelAfter = uint32(block.timestamp) + 24 hours;
    emit Created();
}

/// @notice Release ether or token in escrow to the buyer.
/// @return bool
function release() external onlySeller returns (bool) {
    transferEscrowAndFees(buyer, amount, fee);
    emit Released();
    return true;
}

/// @notice Transfer the value of an escrow
/// @param _to Recipient address
/// @param _amount Amount to be transfered
/// @param _fee Fee to be transfered
function transferEscrowAndFees(address _to, uint256 _amount, uint32 _fee) private {
    withdraw(_to, _amount);
    if (_fee > 0) {
        withdraw(feeRecipient, _fee);
    }
}

/// @notice Cancel the escrow as a buyer with 0 fees
/// @return bool
function buyerCancel() external onlyBuyer returns (bool) {
    transferEscrowAndFees(seller, amount + fee, 0);
    emit CancelledByBuyer();
    return true;
}

/// @notice Cancel the escrow as a seller
/// @return bool
function sellerCancel() external onlySeller returns (bool) {
    if (sellerCanCancelAfter <= 1 || sellerCanCancelAfter > block.timestamp) {
        return false;
    }

    transferEscrowAndFees(seller, amount + fee, 0);
    emit CancelledBySeller();
    return true;
}

/// @notice Disable the seller from cancelling
/// @return bool
function markAsPaid() external onlyBuyer returns (bool) {
    sellerCanCancelAfter = 1;
    emit SellerCancelDisabled();
    return true;
}

/// @notice Withdraw values in the contract
/// @param _to Address to withdraw fees in to
/// @param _amount Amount to withdraw
function withdraw(address _to, uint256 _amount) private  {
    if (token == address(0)) {
        (bool sent,) = payable(_to).call{value: _amount}("");
        require(sent, "Failed to send MATIC");
    } else {
```

```
            require(IERC20(token).transfer(payable(_to), _amount), "Failed to send
tokens");
        }
    }

    /// @notice Allow seller or buyer to open a dispute
    function openDispute() external {
      require(_msgSender() == seller || _msgSender() == buyer, "Must be seller or buyer");
      require(sellerCanCancelAfter > 0, "Funds not escrowed yet");

      dispute = true;
      emit DisputeOpened();
    }

    /// @notice Allow arbitrator to resolve a dispute
    /// @param _winner Address to receive the escrowed values - fees
    function resolveDispute(address payable _winner) external onlyArbitrator {
      require(dispute, "Dispute is not open");
      require(_winner == seller || _winner == buyer, "Winner must be seller or buyer");

      emit DisputeResolved();
      transferEscrowAndFees(_winner, amount, fee);
    }

    /// @notice Version recipient
    function versionRecipient() external pure returns (string memory) {
                return "1.0";
       }
}
```

The contract is initialized with a constructor function, which takes several parameters: the buyer's address, the address of the digital asset being escrowed (or 0x0 if the asset is the blockchain native token), the amount of the asset being escrowed, a fee, the address of the arbitrator, the address of the party who will receive the fee, and the address of a trusted forwarder.

The contract has several modifier functions (onlySeller, onlyArbitrator, onlyBuyer) which are used to ensure that certain functions can only be called by the seller, arbitrator, or buyer, respectively.

The escrow() function allows the seller to fund the escrow with the specified amount of digital asset and fee.

The release() function allows the seller to release the escrowed digital asset to the buyer.

The buyerCancel() function allows the buyer to cancel the escrow without paying a fee.

The sellerCancel() function allows the seller to cancel the escrow after a specified period of time (default 24 hours) has passed, with the seller paying a fee.

The openDispute() function allows either the buyer or the seller to open a dispute, which can be resolved by the arbitrator using the resolveDispute() function.

The transferEscrowAndFees() function is a private function used to transfer the escrowed digital asset and the fee to the specified recipients.

The contract also contains several events that are emitted in response to certain actions being taken, such as Created, Released, CancelledByBuyer, CancelledBySeller, DisputeOpened, and

DisputeResolved, which can be used to trigger other actions in dependent contracts, or to be logged by external services.

The escrow() function can only be called by anyone. The function also checks that the sellerCanCancelAfter variable is equal to 0, meaning that the funds haven't been escrowed yet. The function is able to receive the native token or pull the ERC20 token from the seller to the contract.

The release() function can only be called by the seller, this is done by using the onlySeller modifier. This function takes the escrowed asset and fee and transfers it to the buyer, also it emits the Released event.

The buyerCancel() function can only be called by the buyer, this is done by using the onlyBuyer modifier. This function releases the asset and fee back to the seller and emits the CancelledByBuyer event

The sellerCancel() function can only be called by the seller, this is done by using the onlySeller modifier. The function checks that the current timestamp is greater than the sellerCanCancelAfter variable, meaning that enough time has passed since the funds were escrowed, it also transfers the asset and fee back to the seller and emits the CancelledBySeller event.

The openDispute() function can be called by both the buyer and the seller (there's no modifier being used). This function sets the dispute variable to true and emits the DisputeOpened event.

The resolveDispute() function can only be called by the arbitrator, this is done by using the onlyArbitrator modifier. This function can transfer the asset and fee to either the buyer or the seller, based on the arbitrator's decision and sets the dispute variable to false and emits the DisputeResolved event.

By using the onlySeller and onlyBuyer modifiers, the contract ensures that certain functions (like release() and buyerCancel()) can only be called by the seller or the buyer, respectively. This means that the arbitrator, or any other unauthorized address, would not be able to call these functions and release the funds without the proper authorization.

The openDispute() and resolveDispute() functions are meant to be used by the buyer, seller and the arbitrator. The openDispute() function allows either the buyer or the seller to open a dispute, and the resolveDispute() function allows the arbitrator to resolve the dispute by transferring the funds to either the buyer or the seller. By using the onlyArbitrator modifier on the resolveDispute() function, the contract ensures that only the arbitrator address specified in the constructor can resolve the dispute and transfer the funds, preventing unauthorized access by other addresses.

The transferEscrowAndFees() is a private function that is used internally by the contract to transfer the assets and fee. This function is not visible or accessible outside the contract, and thus can't be called by the arbitrator or any other external address.

The contract also has a state variable dispute which is true when a dispute is open, this ensures that the contract only allows the arbitrator to resolve the dispute and transfer the funds when a dispute is open. If a dispute is not open, the resolveDispute function will not allow to execute the fund transfer.

By combining these different mechanisms, the contract provides a multi-layered protection to prevent unauthorized access to the funds, and ensures that the funds can only be transferred to the proper recipients (buyer or seller) as authorized by the arbitrator during the dispute resolution process.

OpenPeer Adoption

OpenPeer isn't limited to a simple peer-to-peer trading platform like those offered by centralized exchanges today. In the long term we want to compete with the largest onramps like Moonpay and Wyre, bringing mass adoption to crypto. We believe peer-to-peer trading unlocks crypto in emerging markets where it is needed the most. OpenPeer will look to integrate directly with wallets, DeFi protocols, NFT marketplaces and future platforms with the mission to improve crypto onboarding for users in all emerging markets.

Self-Custody Wallets

OpenPeer will be integrated as an onramp option in self-custody wallets both on desktop and mobile. Many of the popular wallets today like Metamask and Argent only have onramp options that are compatible with users from western markets. When using these onramps many users have their payments blocked or are unable to pass verification. For most jurisdictions onramp options are simply unavailable. OpenPeer will enable self-custody wallets to onboard users from any country globally through P2P.

Decentralized Applications and Games

Decentralized applications and games today often have a portal where users of the game can buy crypto to get started. OpenPeer will integrate with these applications as a way for users to buy major cryptocurrencies as well as the applications native tokens. As an application developer, you can only sell your token through traditional onramps after reaching a certain size and market capitalization. For new developers they are forced to have users onramp into ETH or MATIC and then have users exchange to the native token through a decentralized exchange. Rather than having users go through multiple steps, applications can provide liquidity for their native token or have users buy stablecoins with OpenPeer automatically converting USDC or USDT to the native token after the trade.

Token Mechanisms

Buyback Mechanism

At token launch 20% of all trading fees will be used to buyback tokens from the open market and distribute them to token stakers in a staking pool. This presents an efficient way to incentivize and loyalty across a wide selection of stakeholders including users, token holders and arbitrators.

Arbitration Mechanism

The core mechanism of the OpenPeer token will be for the chance to arbitrate disputes. Token stakers can further stake their tokens for the opportunity to resolve disputes between users. Fees earned from

disputes will be paid out to arbitrators in the token used to pay the fee. Currently arbitration fees need to be paid in the native token where the trade is taking place, for example MATIC on Polygon. In the future arbitration fees will be able to be paid in a wider variety of popular tokens. Arbitration is further detailed in the following section.

Governance

Token stakers will be governors of the OpenPeer protocol. Some key governance decisions will be fees, burn rates and which chains OpenPeer will be deployed on in the future. A simple delay mechanism will be implemented to prevent just-in-time voting. Arbitrators who have further staked their tokens can also participate in governance.

Insurance

While chargebacks are less common with bank transfers compared to credit and debit cards they do still exist in P2P trades. In the future, crypto sellers may be able to pay an additional fee to insure their trades with verified buyers and through integrated payment methods. In the event of a chargeback they will have their fiat funds refunded by token stakers who further stake their tokens in an insurance module in return for additional yield.

OpenPeer Arbitration

In the long term, arbitration is a key element in the OpenPeer protocol and its decentralization. At launch, disputes will be arbitrated by the OpenPeer team through a multisig wallet. However over time the arbitration will be decentralized to the community.

Arbitration Token Mechanics

OpenPeer’s arbitration will be powered by the protocol token. Arbitrators have an economic interest to arbitrate decisions as they will collect dispute fees in exchange for their work. By staking tokens, arbitrators can be selected in a pool and vote on how to resolve the dispute. The probability of being selected as an arbitrator on a specific case is proportional to the amount of tokens staked. The higher the amount of tokens staked, the higher chance of being assigned as an arbitrator to a dispute and potentially earning fees. The protocol token will also be used for governance to manage the treasury and set protocol parameters like fees and available tokens.

The protocol token prevents sybil attacks with decentralized arbitration. If arbitrators were simply drawn randomly, someone could create multiple wallets to be drawn multiple times in a dispute and fully control the outcome. The protocol token also incentivizes engagement and honesty. If an arbitrator has tokens staked and doesn’t participate in resolving a dispute they could lose part of their stake. Similarly if they vote incoherently with other arbitrators, they could also lose part of their stake to coherent voters.

The token and arbitration mechanism will evolve over time and may allow for delegation and fee-sharing.

Arbitrator Selection

When decentralized arbitration is launched, there will only be a single staking pool that will arbitrate all trades globally. An initial selection of arbitrators will be qualified from frequent buyers and sellers of the protocol. However as the protocol grows, arbitration will be open to all stakers and pools will eventually be divided by countries and payment networks to enable arbitrators to specialize in specific payment networks which they may be more familiar with.

Once arbitrators have staked their protocol tokens in a pool, they can be available to be selected for a dispute. Their votes in the dispute will be weighted based on the number of tokens they have staked up to a maximum amount. Arbitration selection will also make use of random numbers drawn from blockhashes of recent blocks on Polygon or Ethereum.

Arbitration Voting

Once selected for a dispute, arbitrators will have 24 hours to vote on a resolution otherwise they risk losing part of their stake. Votes will not be public until every arbitrator has cast a vote or the time limit has been reached along with a quorum of votes. Not knowing the other arbitrator's vote is important in finding the truth as detailed in Thomas Schelling's Schelling Point, otherwise described as a focal point. A Schelling Point is each person's expectation of what the other expects him to expect to be expected to do. By incentivizing each arbitrator with fees if they vote with the majority, along with not revealing any votes until every party has voted, we can expect arbitrators to vote for the right and honest resolution.

A minimum number of arbitrators participating will be required in each dispute. If a vote fails to reach quorum new arbitrators will be drawn until enough votes have been cast. The minimum number of arbitrators will be decided by a governance vote and is expected to grow as the protocol is increasingly adopted.

Protecting Against Attacks

There are two possible attacks that could be undertaken by a bad actor. The first is controlling the majority of the liquid token supply. This is unlikely as if the protocol grows popular enough where an attack would be attractive, an attacker would need to purchase a significant amount of protocol token driving up the price and losing any economic benefit. OpenPeer may choose to limit transaction sizes based on protocol token liquidity and price in order to reduce the economic incentive for this type of attack.

Another possible attack is to bribe a voting majority of arbitrators. By bribing 51% of arbitrators a bad actor could dictate the outcome of a vote. OpenPeer may introduce an appeals mechanism where arbitrators will be redrawn and the minimum number of voters will double in size. The bad actor would then need to effectively bribe three times the number of arbitrators drastically reducing their economic incentive.

Conclusion

We have briefly introduced OpenPeer, a decentralized peer-to-peer (P2P) protocol for people to exchange crypto assets like Ethereum and USDT (Tether) for fiat money without centralized counterparty risk through escrow smart contracts. The protocol is governed by protocol token holders and disputes are arbitrated by a decentralized community of token stakers. The sustained currency depreciation seen through many developing countries without stable financial systems demonstrates the need for crypto. The collapse of FTX and the failure of centralized onramps to onboard the billions of people in greatest need of better money underscores the need for a decentralized peer-to-peer trading protocol.