Flash Protocol

Release 0.1.0

XIO Network

Feb 20, 2021

GETTING STARTED

Welcome! The pages that follow contain comprehensive documentation of the **Flash Protocol**. If you are new to **Flash**, you might want to check out the *Protocol Overview* or the *FAQ*

CHAPTER

ONE

GETTING STARTED

Explore the sidebar to find more specific documentation covering other aspects of the protocol.

CHAPTER

TWO

DEVELOPER LINKS

The Flash codebase is comprised of an ecosystem of open source components.

- Flash Protocol
- Flash Receiver
- Flash Frontend

CHAPTER

THREE

CONTENTS

Keyword Index, Search Page

3.1 Protocol Overview

Existing DeFi projects aim to provide annualized earnings for specific currencies using smart contracts. Usually once the user funds are successfully allocated in the smart contract, earnings are calculated every Ethereum block (i.e. every ~15 seconds).

So, if the **APY** (Annualized Percentage Yield) is **10%** and a user stakes **$100**, after 1 year (if the APY was stable), he will be able to withdraw **$110**.

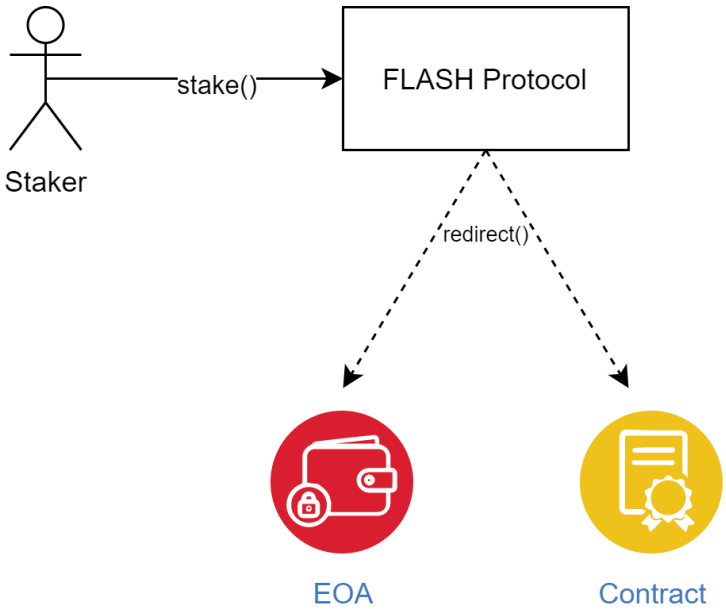
Flash is a permissionless protocol allowing everyone to stake **$FLASH** and earn instant upfront yield.

Flash protocol enables users to stake tokens and earn the yield immediately. Using the same example as above, if the user stakes **$100** for 1 year, he will get **$10** instantly and have his **$100** locked for 1 year.

Because of its permissionless nature, it will exist for as long as Ethereum does.

The Flash Protocol is fueled by the **$FLASH** Token the same way Ethereum is fueled by **ETH**. To learn more about it read *Flash Token*

3.1.1 Features



The Flash Protocol is built with numerous useful features:

- permissionless
- open source
- stake **$FLASH** and redirect the yield to EOA or a contract
- unstake **$FLASH** after the expire period is over
- unstake **$FLASH** before the expire period is over, but burn percentage of the staked amount based on the remaining time
- dynamic FPY (Flash Percentage Yield) calculated with the formula:

$$(1 - \frac{\text{total\_staked\_amount} + \text{new\_staked\_amount}}{\text{total\_supply}}) \div 2$$

- dynamic instant yield calculated with the formula:

$$\frac{\text{new\_staked\_amount} \times \text{expiration\_in\_seconds} \times \text{FPY}}{86400 \times 365}$$

3.1.2 Glossary

Automated market maker

An automated market maker is a smart contract on Ethereum that hold on-chain liquidity reserves. Users can trade against these reserves at prices set by an automated market making formula.

FPY

Flash Percentage Yield - the percentage used to calculate the yield reward.

ERC20

ERC20 tokens are fungibile tokens on Ethereum. Uniswap supports all standard ERC20 implementations.

Pool

Liquidity within a pair is pooled across all liquidity providers.

Liquidity provider / LP

A liquidity provider is someone who deposits an equivalent value of ERC20 tokens into the liquidity pool within a pair. Liquidity providers take on price risk and are compensated with fees.

Slippage

The amount the price moves in a trading pair between when a transaction is submitted and when it is executed.

3.2 FAQ

3.2.1 Basics

What is the Flash Protocol?

Flash is a permissionless protocol allowing everyone to stake **$FLASH** and earn instant upfront yield. Because of its permissionless nature, it will exist for as long as Ethereum does.

How is the Flash Protocol useful?

The main benefit for the users is that they are not required to wait in order to get their yield. Instead the yield is in their possession immediately. On the other hand, the yield can be redirected to either an externally owned account or a contract. This opens infinite amount of possibilities - from redirecting the instant yield to a friend or family member to a complex smart contract that can do further operations with it.

Is Flash Protocol fully permissionless?

Yes. The Flash Protocol Pools/Applications cannot be censored or whitelisted. Users cannot be censored or whitelisted. The XIO Team does not have the power to halt or edit the smart contracts in any way after they've been deployed. The contracts are not upgradeable, and there is no "backdoor" present in the code. Of course, XIO has no control over the contracts of ERC20 tokens placed in the Flash Protocol pools/applications. If a centralized token (e.g., USDC) were to blacklist an address or freeze all transfers, that would affect all USDC tokens everywhere, including those in the Flash Protocol.

Does Flash Protocol charge any fees at the protocol layer?

The users of the Flash Protocol will not be charged any fees. Instead, a percentage of the upfront yield will be "matched" and sent to a predefined Ethereum wallet. The match ratio can be as low as 0% and as high as 20%. That ratio can be changed via a 3 days time-lock function, so users can be aware if a change in the ratio is about to happen.

Example: if the match ratio is 2% and the generated upfront yield is **100 $FLASH**.
100 $FLASH will be send to the user and **2 $FLASH** will be generated for the XIO Foundation. The matched yield will be used to support future developments of The Flash Protocol.

Is there a Flash Protocol token?

Yes, there is a token called $FLASH. The Flash Token does not have a fixed supply and the contract address is **0xb4467e8d621105312a914f1d42f10770c0ffe3c8**. Users can use the $FLASH token in order to stake and earn instant upfront yield.

3.2.2 Miscellaneous

Does using Flash Protocol generate taxable events?

We cannot provide tax or accounting advice. Tax regulations are specific to jurisdiction where you or your company reside. For any legal or tax matters we recommend consulting your own attorney.

Are there risks in using Flash Protocol?

Flash Protocol smart contracts have been designed with security as a top priority. The core protocol code has been reviewed and audited by Solidify (of course, we cannot guarantee that bugs won't be found in the future.)

The Flash Protocol contracts are not upgradeable (though other third-party Pool implementations might be), and there aren't any backdoors.

Remember that the tokens held in any of the Flash Applications are also smart contracts - not controlled by Flash and may have their own risks. The Flash Protocol does not support non-ERC20-conforming tokens, but pools/applications may have been created that use them anyway.

3.3 Flash Token

Flash Token (**$FLASH**) is an Ethereum token that enables interacting with the Flash Protocol.

FLASH contract address is: **0xb4467e8d621105312a914f1d42f10770c0ffe3c8**

The Flash Token does not have a fixed supply. **$FLASH** is minted on every stake and it's burned on every unstake (only if the staking period has not been elapsed). The process of minting can only be achieved using the Flash Protocol.

The inflation rate of the token is corelated to the usage of Flash protocol, the **FPY** (Flash Percentage Yield) and the matching ratio.

The token follows the ERC20 standard with few extensions:

- ERC-3009 - transfer with authorization, which enables gasless transactions
- an exposed mint() and burn() functions which can be invoked only by the flash protocol

3.4 Flash Stake

Flash stakes are a way to stake your $FLASH for certain amount of time and get upfront yield instantly.

For end-users, staking is intuitive: a user picks an amount and staking time and the protocol calculates how much yield they'll receive. The user can also pick the destination of the yield - can be any valid Ethereum address.

If the yield destination is EOA (externally owned account), then the yield is simply transferred to this address. However, if the yield destination is a contract, then a specific contract interface is invoked. This opens a lot of different use cases and opportunities. The Flash Protocol does not know and does not care what happens in the destination contract.

3.4.1 Anatomy of a stake

At the most basic level all stakes in the Flash Protocol happen with a single function, aptly named *stake*

```
function stake(uint256 amountIn, uint256 expiry, address receiver, bytes calldata␣
↪data)
```

- **amountIn:** this is the $FLASH amount that the user is going to stake.
- **expiry:** the time after which the stake will expire.
- **receiver:** the address of the yield destination - externally owned accounts or a contract.
- **data:** arbitrary data passed to the receiving contract - it's up to the dApp what kind of data is passed.

To learn more about the flash receiver, please refer to *Flash Receiver*

3.5 Flash Receiver

Whenever a user stake he can choose the destination of the yield. The yield destination, can be either EOA (externally owned account) or a contract.

If the yield destination is EOA (externally owned account), then the yield is simply transferred to this address.

However, if the yield destination is a contract, then a specific contract interface is invoked.

This functionality allows anyone to build their own dApp with their own flash receiver on top of the Flash Protocol.

3.5.1 Anatomy of the receiver

If the receiver is a contract, the following interface should be exposed by that contract.

```
function receiveFlash(
    bytes32 id,
    uint256 amountIn,
    uint256 expireAfter,
    uint256 mintedAmount,
    address staker,
    bytes calldata data)
external returns (uint256);
```

- **id:** this is the stake id. With the stake id you can fetch the stake data from the protocol.
- **amountIn:** this is the $FLASH amount that the user has staked.
- **expireAfter:** the time after which the stake will expire.
- **mintedAmount:** the amount of minted $FLASH - this is basically the yield being redirected to the Flash Receiver.
- **staker:** the address of the user.
- **data:** arbitrary data passed from the protocol - it's up to the dApp what kind of data is passed.

3.6 Addresses

The contracts are currently deployed on the following networks:

3.6.1 Mainnet

Contract	Address
Flash Token	0xb4467e8d621105312a914f1d42f10770c0ffe3c8
Flash Protocol	0x
Flash App	0x

3.6.2 Rinkeby

Contract	Address
Flash Token	0x
Flash Protocol	0x
Flash App	0x

3.6.3 Ropsten

Contract	Address
Flash Token	0xb4467e8d621105312a914f1d42f10770c0ffe3c8
Flash Protocol	0x
Flash App	0x

3.7 Flash Protocol

See also:

"Smart contracts are pretty difficult to get right." Emin Gün Sirer.

Given the above quote, smart contracts are computer code that define how money moves, so we put the security and simplicity as top priority while building the Flash Protocol.

The Flash Protocol contract includes the core functionalities to perform a flash stake, along with interfaces to fetch all data that you'll ever going to need.

3.7.1 State functions

Those are the functions that require transacting on the Ethereum network.

Stake

```
function stake(uint256 amountIn, uint256 days, address receiver, bytes calldata data)␣
↪external returns (uint256 mintedAmount, uint256 matchedAmount, bytes32 id);
```

Unstake

```
function unstake(bytes32 id) external returns (uint256 withdrawAmount);
```

Unstake Early

```
function stake(uint256 amountIn, uint256 days, address receiver, bytes calldata data)␣
↪external returns (uint256 mintedAmount, uint256 matchedAmount, bytes32 id);
```

3.7.2 View functions

Those are the functions that can be used to fetch data from the protocol.

Get Stake Balance

```
function balances(address staker) external view returns (uint256);
```

Get FPY ratio

```
function getFPY(uint256 amountIn) external view returns (uint256);
```

Get Mint Amount

```
function getMintAmount(uint256 amountIn, uint256 expiry) external view returns␣
↪(uint256);
```

Get Stake By ID

```
function stakes(bytes32 id) external view returns (uint256 amountIn, uint256 expiry,␣
↪uint256 expireAfter, uint256 mintedAmount, address staker, address receiver);
```

3.8 Events

A list of all events emitted by the Flash Protocol

3.8.1 Staked

```
event Staked(
        bytes32 id,
        uint256 amountIn,
        uint256 expiry,
        uint256 expireAfter,
        uint256 mintedAmount,
        address indexed staker,
        address indexed receiver
    );
```

- **id:** this is the stake id. With the stake id you can fetch the stake data from the protocol.
- **amountIn:** this is the $FLASH amount that the user has staked.
- **expiry:** the time after which the stake will expire (e.g. 86400 = 1 expiration)
- **expireAfter:** the block timestamp after which the stake will expire.
- **mintedAmount:** the amount of minted $FLASH - this is basically the yield being redirected to the Flash Receiver.
- **staker:** the address of the user.
- **receiver:** the address of the yield destination.

3.8.2 Unstaked

```
event Unstaked(bytes32 id, uint256 amountIn, address indexed staker);
```

- **id:** this is the stake id. With the stake id you can fetch the stake data from the protocol.
- **amountIn:** this is the $FLASH amount that the user has staked.
- **staker:** the address of the user.

3.8.3 Subgraph

This section explains everything you need to know about the Flash Protocol Subgraph. The Flash Protocol Subgraph listens for events from one or more data sources (Smart Contracts) on the Ethereum Blockchain. It handles indexing and caching of data which can later be queried using an exposed GraphQL API, providing an excellent developer experience.

The Flash Protocol Subgraph is powered by The Graph

See also:

The Graph is a protocol for building decentralized applications (dApps) quickly on Ethereum and IPFS using GraphQL.

Data Sources

- Flash Protocol - <TODO: FLASH_PROTOCOL_ADDRESS>

Resources

- Flash Protocol Subgraph Explorer: <TODO: THE_GRAPH_URL>
- Flash Protocol Subgraph Source: <TODO: GITHUB_URL>

Queries

This page provides various query examples. You can test any of the queries, or write your own, on the Flash Protocol Subgraph Explorer.

- TBD_1
- TBD_2

3.9 Gas Cost

The gas usage of the protocol functions may fluctuate by market and user. Here you can find the typical gas costs per function.

3.9.1 Flash Protocol

Function	Typical Gas Cost
stake	~TBD
unstake	~TBD
unstakeEarly	~TBD

3.9.2 Flash Token

Function	Typical Gas Cost
transfer	~TBD
transferFrom	~TBD
approve	~TBD
permit	~TBD
transferWithAuthorization	~TBD

3.10 Building Flash Receiver

In the example below you can find a simple example for a flash receiver that can be used by the Flash Protocol.

We'll define a function required by the Flash Protocol called `receiveFlash` and we'll simply collect the yield and reward the staker with half of the yield amount in DAI. So if the user stakes **1000 $FLASH** with **10% FPY**, we'll receive **100 $FLASH** and we'll send him **50 DAI**.

Note: The code below is used **only** for example and testing. Do not use on production.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.6.0;

import '@openzeppelin/contracts/math/SafeMath.sol';
import './interfaces/ERC20.sol';

contract FlashReceiver {
    using SafeMath for uint256;

    address public constant FLASH_PROTOCOL =␣
↪0x2Fec418941c8E710372625Df48F32b69Cf4431A9;
    address public constant DAI_ADDRESS = 0x6b175474e89094c44da98b954eedeac495271d0f;

    modifier onlyFlashProtocol() {
        require(msg.sender == FLASH_PROTOCOL, 'Only FlashProtocol can call this');
        _;
    }

    function receiveFlash(
        uint256 amountIn,
        uint256 expireAfter,
        uint256 mintedAmount,
```

(continues on next page)

(continued from previous page)

```
        address staker
    ) public override onlyFlashProtocol returns (uint256) {
        uint256 daiReward = mintedAmount.div(2);
        IERC20(DAI_ADDRESS).transfer(staker, daiReward);
        return daiReward;
    }
}
```

Now, if we deploy the contract we can use the contract address in the `receiver` field when calling `stake` and we'll redirect the user's yield to that contract. The result would be that the user will stake **$FLASH** and receive **$DAI**.

3.11 Security

Flash Protocol smart contracts have been designed with security as a top priority. The core protocol code has been reviewed and audited by Solidify (of course, we cannot guarantee that bugs won't be found in the future.)

- Audit result: https://github.com/solidified-platform/audits/blob/master/Audit%20Report%20-%20Flash%20Protocol%20%5B04.12.2020%5D.pdf

3.12 Getting started

So, you know the basics, let's set everything up to code! To start developing dApps on top of The Flash Protocol, you can use the SDK we've built.

3.12.1 1. Installing

```
yarn install <TODO: package_name>
```

2. Importing

```
import {  } from "package_name";
```

3.13 Fetching protocol data

3.14 Stake

3.15 Unstake