# LIFT
# Whitepaper v1.1.0

The platform token for the LIFTOFF.eth token launchpad

*Lid Protocol Team, 2021.*

## 1. Overview

LIFTOFF.eth provides the most advanced token sale buyer protections to date. These protections include required informational disclosures, smart contract token distribution, automated locked liquidity, and limited crash insurance. LIFT adds the final missing a critical component: community. Successful token sales require an active and vigilant community who are rewarded for successful launches. The LIFT platform token aims to spark community through rewards, memberships, and voice.

The LIFT token will begin with several core benefits for whitelisted LIFT members. First, every token sale will airdrop 3% of supply proportionately to members. Second, members gain access to private chat rooms and channels for reviewing and discussing LIFTOFF sales. Third, members publicly verify LIFTOFF launches as a signal to the wider DeFi community. Combined these benefits encourage and support an active community. The LIFT sale, hosted on LIFTOFF, will be open to the public with only unfriendly jurisdiction limitations. After launch, the token will be freely tradeable on Uniswap.

LIFT membership has tighter restrictions than most other platform tokens. To join, members must hold 1 LIFT, verify ownership of a discord account, and receive whitelisting from the Lid Protocol team at its discretion. Members must maintain a 1 LIFT balance at all times. Whitelisting and balance maintenance requirements allow the removal of members who take advantage of the community. These misbehaviors include but are not limited to spreading disinformation, exploiting benefits, running unapproved bots, or otherwise causing measurable harm to the LIFT community.

## 2. Background

Lid Protocol's first platform, Lid Bonding Curve Sales, successfully sold LID tokens and locked liquidity. However, the complexity of bonding curve technology confused most users. Lid Simplified superseded the bonding curve platform. Lid Simplified had a lifespan of 6 months with dozens of successful sales. Lid Simplified prevented Uniswap liquidity removals through the automated locking of liquidity in a fully decentralized, trustless fashioned. However, Lid Simplified was vulnerable to "Soft

Exit" attacks where the project team would delete social media channels and websites immediately after a sale concluded.

LIFTOFF.eth retains all of Lid Simplified's advantages while adding protections against Soft Exit attacks. Just like Lid Simplified, LIFTOFF provides automated liquidity locking, ETH distribution, and token distribution. However, it has several fundamental improvements.

First, LIFTOFF does not allocate the project developers any tokens. Fair allocation, meaning 100% of tokens to liquidity and users, removes the volatility from insider sales. To acquire tokens, project developers must allocate some amount of their ETH payment to purchasing tokens. This provides provides price stability and reduces volatility risks.

Second, LIFTOFF provides crash insurance for 10 weeks. During the first week crash insurance covers 100% of the available tokens. Excess insurance redemptions unwind the sale, preventing the project developers from receiving any payment. The crash insurance gradually covers a lower percentage of supply of the next 10 weeks, and redemptions are taken from the project developer's ETH. Thus Crash Insurance is fully secured by the sale itself. As the project developer pay is 10% per week for 10 weeks, the project developer is incentive to continue active involvement. Time delays also create a high effort barrier, albeit not perfect, to scammers.

Third, LIFTOFF utilizes XLOCK technology with XETH to increases the ETH allocation to 110%. XLOCK, forked from ULOCK and ROOTKIT, frees up ETH from locked liquidity. This 10% boost increases liquidity and allows 110% of the total ETH raise to be allocated instead of 100%. However, since the default Uniswap UI does not recognize XETH as a routing pair, traders must use the Penguinswap frontend instead.

The LIFTOFF platform is already alive and active on the Ethereum Mainnet. Once the LIFT token is released, the rewards will begin immediately. While future rewards, benefits, and upgrades may be developed, buyers of the LIFT token should purchase with the understanding that the platform is fully developed, operational, and complete as is.

## 3. The LIFTOFF platform

The LIFTOFF platform is a fully decentralized, autonomous, self service platform. By fully decentralized, we mean the LIFTOFF platform cannot be censored or managed by any centralized authority. The domain LIFTOFF.eth is hosted on ENS, a fully decentralized domain service. The dapp files are hosted on IPFS, a decentralized hypermedia protocol. LIFTOFF's smart contracts are deployed on the Ethereum Mainnet as well as Ropsten.

The Lid Protocol Team has the rights to update the dapp, point the ENS domains to new IPFS sites, and upgrade the LIFTOFF smart contracts. However, the normal day-to-day operation of LIFTOFF does not require any efforts by the Lid Protocol Team.

New launches can be created by any dev at any time by just filling out a form and sending an Ethereum transaction. Accordingly, users of the LIFTOFF platform need to conduct their own research and exercise caution in which sales they participate in. To aid this process, the Liftoff launchpad has both required and optional fields which project devs fill to provide information about their token sale. Buyers of tokens on the LIFTOFF platform are then able to exercise discretion both from the information given and request explanation from the project devs for missing information. Tokens are encouraged to already have a live, working dapp as one of the fields provided in the launchpad. Lack of a live, working dapp for a token sale should act as a strong warning sign to LIFTOFF users as all

LIFTOFF tokens present themselves as utility tokens, not securities. Investigation of deceit around the nature of a token may create a strong warning sign not to purchase a particular token.

The LIFTOFF platform is restricted in several countries, particularly the USA. As of 2021 there remains substantial regulatory uncertainty around token sales to US citizens, residents and representatives. The LIFTOFF platform is unable to geoblock IP addresses. The dapp is delivered via IPFS and the smart contracts run on Ethereum with no backend services, so no central server can be configured to block IP addresses. However, the LIFTOFF platform reminds token sale buyers that they must be in a crypto friendly jurisdiction to use the dapp. Each time the Ignite button on a dapp is clicked, before the transaction is sent the prospective token buyer is required to verify they are not in the USA or other unfriendly jurisdiction.

Every launch on LIFTOFF has the same core tokenomics, but may have different token quantities. See section “LIFT Tokenomics” for a sample. One important fact to note on the liquidity pools is 40% is in ETH/xxx and 19.91% is in LID/xxx, where xxx is the launched token. Since the LID for the LID/xxx pair is purchased from the raised ETH, and the ETH is distributed over 10 weeks, the LID/xxx liquidity pool is created and increased over 10 weeks. Gradually adding the LID/xxx liquidity has two major advantages: (1) Gradual liquidity prevents sudden increases in the LID price from purchases which may be exploited by bad actors, and (2) Gradual liquidity allows faster price movements when the token initially launches, meaning faster price discovery as the token reaches equilibrium. There is a drawback compared to Lid Simplified, the previous legacy platform. As LID/xxx pools are not created in the first week, fees from arbitration bots will not generate staking rewards during this typically high volume trading period.

Like all Lid Protocol projects, LIFTOFF is radically open source. All the smart contracts, deployment scripts, dapps, tooling, and tests are open source with a GPLv3 license. GPLv3 was chosen as a relatively permissive copyleft license, preventing Lid Protocol’s code being ever incorporated into proprietary software. Anyone forking Lid Protocol’s projects must be as radically open source as Lid Protocol is.

## 4. The XLOCK platform

The XLOCK platform, just like LIFTOFF, is fully decentralized with the same meaning of “fully decentralized” as given in the section “The LIFTOFF Platform.” The ENS site hosted on IPFS is available at xlock.eth, and all smart contracts are deployed onto Ethereum. Like LIFTOFF, the Lid Protocol team has the ability to update the dapp, point the ENS domain to new IPFS sites, and upgrade the XLOCK smart contracts excluding the XETH contract. The normal day-to-day operations of XLOCK do not require any effort by the Lid Protocol team, although periodically the xETH/ETH pool managed by the xethLiqManager contract must be updated as described in more detail at the end of this section.

XLOCK is an open, 0 fee platform that allows the creation of infinite liquidity on Uniswap. The technology is forked from uLock which was inspired by rootkit’s ERC-31337. Several upgrades were applied to this standard by uLock to allow the creation of arbitrary tokens against an xEth pool.

The infinite liquidity of XLOCK only exists in a single case; where 100% of tokens are locked as liquidity. In Liftoff’s case, less than 100% are locked as liquidity, so the liquidity bonus provided by XLOCK is limited to 10% of the total sale size.

For all Uniswap pairs, Uniswap uses an invariant $I$ for tokens $X$ and $Y$ s.t. $X*Y=I$. For token contracts with a capped total supply, treading against an ETH pair, this means there is always some

ETH remaining in the contract. This can be quite substantial, sometimes in the 1000's of ETH. XLOCK takes this insight a step further, realizing that if 100% of a tokens liquidity is locked, 100% of the liquidity's ETH is permanently locked. XLOCK thus uses a WETH fork, xETH, to create the pools unbacked. Since the locked xETH can never be redeemed, all circulating xETH is always backed 1:1 by ETH. More information is available on the xlock-react GitHub. The fundamental research is available in the Rootkit whitepaper.

In addition to creating free liquidity, XLOCK also creates an xETH/ETH pool on Uniswap using ETH available in the xETH contract. Since most of the ETH most of the time in the xETH contract is not accessed, a safe percentage of the ETH can be withdrawn to create an xETH/ETH liquidity pool as long as the UNI LP are held in a contract which will withdraw the LP and return the ETH to the xETH contract in case the backing ratio falls too low. As the implication of rebalancing the xETH/ETH pair on arbitration bots is still unknown, the xETH/ETH rebalancing currently requires an authorized member of the Lid Protocol Team to occasionally trigger the function. Once further research demonstrates the method can be safely made public, it will be available for anyone to call.

Just like LIFTOFF and all other Lid Protocol projects, XLOCK is radically open source. All the smart contracts, deployment scripts, dapps, tooling, and tests are open source under GPLv3 license.

## 5. LIFT Features

LIFT will start with Tier 1 requirements and rewards. Tier 1 will always have the best rewards – any future benefits will accrue to Tier 1 first. To apply to be a Tier 1 member, the applicant must hold 1 LIFT and verify their Ethereum address for whitelisting from their discord account. If a member's account ever drops below 1 LIFT, you will lose your Tier 1 status and will need to reapply.

Tier 1:

- Benefits:
  - 3% airdrop from each Liftoff (10 weeks after sale)
  - Private Liftoff discord channel access
  - Vote on verification of Liftoff sales
- Requirements:
  - Maintain balance of 1 LIFT.
  - Good standing within LIFT community.
  - Whitelist address with verification from Discord account.

Tier 2+:

- Tier 1's current benefits will be given to all new tiers.
- Tier 1 will be the first to receive all new benefits.
- Tiers 2+ will have lower LIFT balance requirements.

## 4. LIFT Tokenomics

The LIFT tokenomics are designed to reward all current and future actors in both the Liftoff and Lid ecosystems. LIFT will not supplant the LID token; rather, it complements LID by driving adoption of Liftoff. LID will remain the Liquidity token for Liftoff, and ~20% of each raise in ETH will be converted to LID locked liquidity. All LIFTOFF launches will follow the same tokenomics, except with the addition of a 3% airdrop to eligible LIFT holders.

Token Allocations:

- 2000 LIFT total supply
- 1251 LIFT to ignitors
- 749 LIFT to liquidity

Pricing details:

- Softcap: 100 eth
- Hardcap: 650 eth
- Price at hardcap: 1.92 LIFT/ETH
- If hardcap is not reached, excess tokens are distributed proportionately to buyers.

ETH allocations, as % of raise:

- 59.91% liq
- 5.09% fees
- 30% LIFT buyback&burn
- 10% Liftoff Partners (marketing)
- 5% LIFT buy and airdrop to LID stakers at block number 11729000

# Appendix A. LIFTOFF Contracts

The LIFTOFF platform is composed of 4 primary smart contracts: LiftoffSettings, LiftoffEngine, LiftoffInsurance, and Liftoff Registration. These interfaces and their methods are described in detail below. All Liftoff contracts have been audited by Halborn and are verified on Etherscan.

## Deployed Addresses

Ropsten

```
LiftoffSettings:      0x71A442F174EA408f762624981c07Ca3F800Aa72E
LiftoffEngine:        0xF4CCd4483b393A0526FF939CF15E89eBd3958f2B
LiftoffInsurance:     0x4f250e2236457259BDac9f651f62e972Ce502Caa
LiftoffRegistration: 0xadd1539fb19e03eE66fd365a6F6Cad68b49f981C
```

Mainnet

```
LiftoffSettings:      0xF54d9fC14A006763C83d4e12B1BB5dFB02eA668c
LiftoffEngine:        0x22bCFca3E18B2e7f114e17245e50D9dBf8Bb5e47
LiftoffInsurance:     0x4013F4366beEccb04d2462b061b0C78499eE8FfD
LiftoffRegistration: 0xA4Bf30C2f474fB2ffa35dBF432142DEa08D6655E
```

## Interface Documentation

**LiftoffSettings**

LiftoffSettings provides properties to be read by other Liftoff contracts. These can be set by a governance address only. All methods use a get/set pattern for their properties.

function setEthXLockBP(uint _val) external;
function getEthXLockBP() external view returns (uint);
The basis points of raise that will be locked through xLocker.

function setTokenUserBP(uint _val) external;
function getTokenUserBP() external view returns (uint);
The basis points of tokens bought from pool to be distributed as rewards.

function setLiftoffInsurance(address _val) external;
function getLiftoffInsurance() external view returns (address);
Address for the deployed LiftoffInsurance contract.

function setLiftoffLauncher(address _val) external;
function getLiftoffLauncher() external view returns (address);
Address for the deployed LiftoffLauncher contract.

function setLiftoffEngine(address _val) external;
function getLiftoffEngine() external view returns (address);
Address for the deployed LiftoffEngine contract.

function setXEth(address _val) external;
function getXEth() external view returns (address);
Address for the deployed xEth contract.

function setXLocker(address _val) external;
function getXLocker() external view returns (address);
Address for the deployed xLocker contract.

function setUniswapRouter(address _val) external;
function getUniswapRouter() external view returns (address);
Address for the deployed Uniswapv2Router02 contract.
0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D

function setInsurancePeriod(uint _val) external;
function getInsurancePeriod() external view returns (uint);
Time period for 1 Insurance Cycle. Usually 7 days.

function setLidTreasury(address _val) external;
function getLidTreasury() external view returns (address);
Address for the Lid Treasury for receiving fees.

function setLidPoolManager(address _val) external;
function getLidPoolManager() external view returns (address);
Address for the Lid Pool Manager for making Lid/xxx pools.

```
function setXethBP(
    uint _baseFeeBP,
    uint _ethBuyBP,
    uint _projectDevBP,
    uint _mainFeeBP,
    uint _lidPoolBP
) external;
```
function getBaseFeeBP() external view returns (uint);
function getEthBuyBP() external view returns (uint);
function getProjectDevBP() external view returns (uint);
function getMainFeeBP() external view returns (uint);
function getLidPoolBP() external view returns (uint);
BaseFeeBP lid fee is basis points of sale charged even when a sale refunds. EthBuyBP is basis points of sale to purchase xxx from uniswap. ProjectDevBP is basis points of sale granted to dev. MainFeeBP is basis points of sale to Lid Treasury. Lid Pool BP is basis points of sale to purchase Lid for Lid/xxx pools. Sum must be 10000 basis points. ProjectDevBP, MainFeeBP, LidPoolBP, are distributed to their respective recipients over 10 insurance periods discounted by redeemed xxx tokens.

**LiftoffRegistration**

LiftoffRegistration is the entry point for project devs who wish to launch on Liftoff. It registers references to offchain data, and calls the LiftoffEngine.

```
function registerProject(
   string calldata ipfsProjectJsonHash,
   string calldata ipfsProjectLogoHash,
   string calldata ipfsProjectOpenGraphHash,
   uint launchTime,
   uint softCap,
   uint hardCap,
   uint totalSupplyWad,
   string calldata name,
   string calldata symbol
 ) external;
```

Requires that the launchTime is within the window set by minLaunchTime and maxLaunchTime. Requires the total token supply is below an upper limit of 1 trillion. Calls liftoffEngine.launchToken. Stores the ipfsHash of offchain data at tokenIpfsHash[tokenId] where tokenId is returned by launchToken, the index of the token in order of registration.

function setSoftCapTimer(uint _seconds) public;
softCapTimer is the amount of time that the project must reach softcap or be refunded.

function setLaunchTimeWindow(uint _min, uint _max) public;
Time window, in seconds from present, within which the launch time must be set.

function setLiftoffEngine(ILiftoffEngine _liftoffEngine) public;
Address of the LiftoffEngine.

**LiftoffEngine**

LiftoffEngine is the core of Liftoff. It follows the TokenSale from launch until it enters either refund or insurance.

```
struct TokenSale {
   uint startTime;
   uint endTime;
   uint softCap;
   uint hardCap;
   uint totalIgnited;
   uint totalSupply;
   uint rewardSupply;
   address projectDev;
   address deployed;
   bool isSparked;
   string name;
   string symbol;
   mapping(address => Ignitor) ignitors;
```

}
Data structure for TokenSales. startTime: block time at which sale starts. endTime: block time at which sale ends, unless hardcap is reach first. softCap: minimum totalIgnited to reach before endTime or the TokenSale refunds. hardCap: maximum totalIgnited, when hit ends the token sale. totalSupply: total tokensupply. rewardSupply: quantity of tokens available to claim by Ignitors. projectDev: address which will receive the project dev xEth. deployed: address at which the token sale is deployed by xLocker. isSparked: true once spark is called for the token sale. name: Name of the token, consumed by xLocker. symbol: Symbol of the token, consumed by xLocker. ignitors: all ignitors which ignited eth for the TokenSale.

```
struct Ignitor {
   uint ignited;
   bool hasClaimed;
   bool hasRefunded;
}
```
Data structure for Ignitors. ignited: amount of xEth ignited by the Ignitor. hasClaimed: whether the Ignitor has claimed their rewards. hasRefunded: whether the Ignitore has claimed their refund if the token sale failed to reach softCap.

```
function launchToken(
   uint _startTime,
   uint _endTime,
   uint _softCap,
   uint _hardCap,
   uint _totalSupply,
   string calldata _name,
   string calldata _symbol,
   address _projectDev
) external returns (uint tokenId);
```
Creates a new TokenSale. Can only be called by LiftoffRegistration. Assigned a sequential nonce tokenId from totalTokenSales. Passed values correlate to their associated values in the TokenSale struct. Stores the TokenSale in uint[] public tokens.

```
function igniteEth(uint _tokenSaleId) external payable;
```
Same as function ignite() but converts ignited eth to xEth, and refunds eth in excess of hardcap as eth.

```
function ignite(uint _tokenSaleId, address _for, uint _amountXEth) external;
```
Ignites xEth from msg.sender. Refunds xEth in exccess of the hardCap for the TokenSale from tokens[_tokenSaleId]. Adds ignited xEth to the Ignitor at address _for. msg.sender must have approved the LiftoffEngine contract to spend xEth at least for _amountXEth. Can only be run when the TokenSale is igniting. Calculates the amount to ignite with getAmountToIgnite(), then transfers the amount returned from the sender and ignites it, adding the amount to the Ignitor and TokenSale.

```
function claimReward(uint _tokenSaleId, address _for) external;
```
Claims TokenSale rewards for Ignitor from the TokenSale at tokens[_tokenSaleId] and the Ignitor at TokenSale.ignitors[_for]. Can only be run after a TokenSale has been sparked. May only be run once

per Ignitor, with the ignitor.hasClaimed check. Must set this value to true, to prevent multiple claims. Transfers reward, returned by getReward(), to _for.

function spark(uint _tokenSaleId) external;
Sparks the TokenSale. Deploys the token using xLocker, allocates tokens, and registers insurance. Can only be called if isSparkReady is true. Must set tokenSale.isSparked to true, to prevent multiple sparks for one sale. _deployViaXLock calculates the xEthLocked value to launch an ERC20 token via xLocker, then immediately buys xEthBuy amount of tokens, finally saving the deployed address to the TokenSale and returns the xEthBuy amount. _allocateTokensPostDeploy sets the rewardSupply of the tokenSale. _insuranceRegistration registers the TokenSale with insurance, sends the xEth remaining after the uniswap buy to the insurance, and sends the remaining tokens after the rewardSupply to the insurance contract. The Insurance contract is responsible for all distributions except the rewardSupply.

function claimRefund(uint _tokenSaleId, address payable _for) external;
Sends the refund available for _for at the TokenSale to _for. Can only run when isRefunding is true. Can only run when ignitor.hasRefunded, and must set this to true to prevent multiple refunds for the same account. Transfers ignited xEth back to the ignitor.

```
function getTokenSale(uint _tokenSaleId) external view returns (
   uint startTime,
   uint endTime,
   uint softCap,
   uint hardCap,
   uint totalIgnited,
   uint totalSupply,
   uint rewardSupply,
   address projectDev,
   address deployed,
   bool isSparked
);
```
Returns the values of TokenSale stored at tokens[_tokenSaleId]

```
function getTokenSaleForInsurance(uint _tokenSaleId) external view returns (
   uint totalIgnited,
   uint rewardSupply,
   address projectDev,
   address deployed
);
```
Returns the values of TokenSale needed by LiftoffInsurance stored at tokens[_tokenSaleId]

```
function isSparkReady(
   uint endTime,
   uint totalIgnited,
   uint hardCap,
   uint softCap,
   bool isSparked
```

) external view returns (bool);
Checks if TokenSale is available to spark. Always false if isSparked is true. Otherwise, only true if one of the following is true:
totalIgnited is greater or equal to the softCap AND the current block time is past the endTime,
totalIgnited is greater than or equal to the hardCap.

function isIgniting(
uint startTime,
uint endTime,
uint totalIgnited,
uint hardCap
) external view returns (bool);
Checks if TokenSale is currently igniting. Only true if one of the following is true:
The block time is within the startTime to endTime window,
totalIgnited is equal to or greater than the hardcap.

function isRefunding(
uint endTime,
uint softCap,
uint totalIgnited
) external view returns (bool);
Checks if the TokenSale is current refunding. Ony true if both (1) totalIgnited is below the softcap AND (2) now is passed the endTime.

function getReward(
uint ignited,
uint rewardSupply,
uint totalIgnited
) external pure returns (uint reward);
Calculates the amount of reward that would be available given an amount ignited, a rewardSupply, and the totalIgnited in the TokenSale.

**LiftoffInsurance**
LiftoffInsurance does 2 tasks: (1) allows users to redeem tokens for eth at the original price minus the BaseFee (2) distributes tokens and xEth to Lid addresses and the token project dev.

struct TokenInsurance {
uint startTime;
uint totalIgnited;
uint tokensPerEthWad;
uint baseXEth;
uint baseTokenLidPool;
uint redeemedXEth;
uint claimedXEth;
uint claimedTokenLidPool;
address deployed;

```
    address projectDev;
    bool isUnwound;
    bool hasBaseFeeClaimed;
}
```
Data structure for insurance. startTime: Time the first insurance cycle started. totalIgnited: see TokenSale.totalIgnited. tokensPerEthWad: price in tokens per eth times 10^18 that the TokenSale originally launched at. baseXEth: amount insured at 100% for the first week. baseTokenLidPool: Tokens allocated for the Lid/xxx pool. redeemedXEth: total xEth redeemed for insurance. claimedXEth: total claimed xEth. claimedTokenLidPool: total claimed tokens for the Lid/xxx pool. deployed: ERC20 deployed address of the token. projectDev: token's project dev, to receive project dev share of xEth every period. isUnwound: triggers unwind of insurance, where insurance sells all tokens onto market for full refunding. hasBaseFeeClaimed: true once the base fee, which is on all sales, is triggered.

function register(uint _tokenSaleId) external;
Registers the TokenSale from LiftoffEngine as insured. Sets tokenIsRegistered[tokenSaleId] to true. Must be run before createInsurance. Can only be called by LiftoffEngine.

function redeem(uint _tokenSaleId, uint _amount) external;
Gives xEth at initial presale rate to sender for tokens. Insurance  must be first initialized. Redeemer must first approve _amount of tokens. If the redeem causes the insurance to excess the baseXEth, reverts if after first period. Otherwise, triggers unwind and sells all token held by contract on uniswap. If so, must set tokenInsurance.isUnwound to true.

function claim(uint _tokenSaleId) external;
Distributes xEth and tokens to Lid and project dev. Only callable once per period (7 days). Insurance must be initialized, and must not have been unwound. Claims the baseFee if it has not been claimed, and if so only claims the baseFee and no other distributions. Remaining distributions can only be done after the first period.

function createInsurance(uint _tokenSaleId) external;
Creates the TokenInsurance. Callable by anyone. Can only be called once. Must be called after regsiter(_tokenSaleId) for the same id. Fetches TokenSale data with liftoffEngine.getTokenSaleForInsurance. Stores TokenInsurance at tokenInsurances[_tokenSaleId]