Typhoon Network, Privacy-centric protocol to transfer funds on Binance Smart Chain

Whitepaper v1.0

Authored by the Typhoon Network team

Overview

- Typhoon is a dApp built on Binance Smart Chain with the goal to establish a privacy-centric system on BSC and bring it to mass adoption
- In a time where everyone leaves a trail of information online behind, it is becoming harder and harder to truly stay anonymous. Due to the public nature of blockchains, this became even more difficult as everything done is permanently recorded and viewable by everyone.
- Typhoon solves this problem by utilizing zkSNARK and zero-knowledge technology to enable private transactions on an otherwise public ledger

Goals

- Become the de-facto staple app on BSC to transfer funds privately
- Provide the most secure and private experience to send funds possible, without jeopardising user experience in the process
- Utilize zero-knowledge technology to build a trustless service that operates without team intervention fully onchain
- Build a self-sustaining ecosystem by utilizing the TYPH token to encourage providing anonymity, while also giving the token value
- Enable privacy for users and partners equally by providing openly documented stable APIs

Non-Goals

- Features that don’t have privacy or ecosystem sustainment in mind such as tokenswaps, money-lending or other DeFi initiatives
- Enable transfers for every token imaginable
- Create a for-profit service or release for-profit features

Technical Overview

Typhoon utilizes zkSNARK technology to enable withdrawable deposits without recording information on-chain that can be later used to identify the origin of a deposit.

The trustless lifecycle of a transfer is as follows:

1. Secret generation

The client (here a user using a javascript connected web3 provider such as Metamask or Binance Smart Chain Wallet) generates 2 random numbers of type bigint, called a *secret* and a *nullifier.* The random numbers are combined and used to generate the eventual *nullifierHash* and *commitment* where *nullifierHash = pedersen(nulliifier)* and *commitment = pedersen(nullifier, secret).* The unhashed combined nullifier and secret then become *a note* that is displayed to the user.

2. Deposit

Upon connecting with the smart contract, the provider sends *commitment* together with the deposit amount to the contract. When not full, the contract accepts the deposit and inserts the *commitment* into an internal Merkle Tree with height = 20 as leaf, and re-calculate the latest root. The previous root is stored in an internal history array.
The users deposit information besides *commitment* never leave the users machine and aren't transmitted to the blockchain.

3. Withdrawal

Upon initiating a withdrawal, the client splits the inputted note back into *secret* and *nullifier*, downloads the deposit history from the smart contract and constructs a local representation of the Merkle Tree, then proceeds to search for the *commitment* in said tree.
If found, the *root*, *nullifierHash* (of the note) and *recipient* are used to create a *proof* that the user does indeed know that there is an unspent deposit contained in the tree. This information is then transmitted to the smart-contract and verified. If the users claim is verified onchain, the deposited funds are released and transferred to the specified *recipient* address.

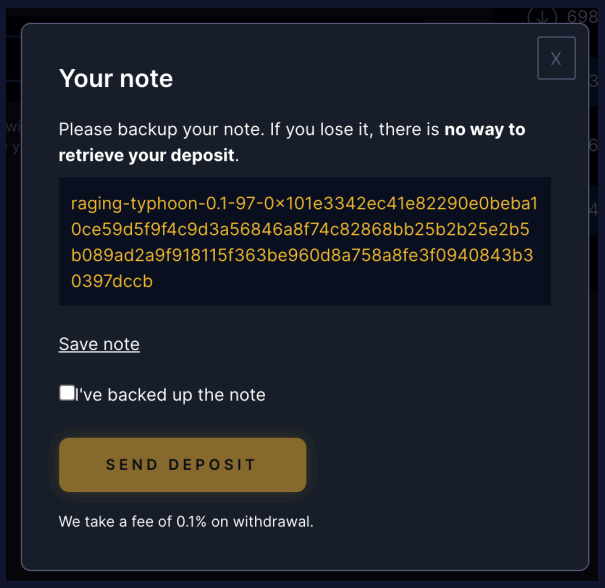
Our Implemented Solution

Using the method outlined above, we provide a an easy-to-use solution in where the client generates a random note without user intervention and presents it with an option to save it to a file. When confirmed, a provider prompt will verify to send the funds and transact the data to the chain.

The note is then given to the user meant to receive the funds.

Upon withdrawal by user 2, the client transparently fetches the deposit information from the Typhoon smart contract, generates the proof and uses the provider to request the releasing of the funds from the smart contract. If the user's claim can be verified on chain, the funds are transferred to the target address.

All information to interact with the contract (including private keys) are available publicly and can be used without the web interface.



Screenshot of the current deposit dialog

How Privacy Is Achieved

All transactions come and go from the Typhoon smart contract and the contract itself only records part of note on chain.
As the note is the only thing linking a withdrawal to a deposit, and this note is not saved on chain, it is not possible to make a direct connection between an incoming transfer and an outgoing transfer.

As the number of deposits increases, any deposit existing on the contract can be the potential source of any given withdrawal.

The Token Ecosystem

Typhoon launches with the TYPH token, our approach to establish longterm sustainability of the service.

The token has the following, but not limited to, use-cases:

1. **Governance** - We want Typhoon to evolve over time and adapt to user needs. To achieve this, the token will be used to draft proposals and vote on how the service should change. The direction of the product will be driven by the community.
2. **Fee Claim** - Typhoon charges a 0.1% fee on withdrawal. Token holders will be able to claim a share of those fees proportional to the amount of tokens they use to interact with the service for staking and governance votings.
3. **Increasing Anonymity** - The token is to be used as rewards for users providing funds to the Typhoon contracts. Provided funds actively help with increasing the anonymity set and securing the usage of other users. To encourage this, TYPH will be attributed to those users.
4. **Fee Reduction -** The token will be used to reduce occurring fees when interacting with the Typhoon smart contracts and provided services.

Token Buyback & Token Burn

To have a constant value and market attached to the token, a regular buyback is to be implemented on-top of the proposed fee-share model. Part of the accumulated fees and locked-up funds are to be used to purchase TYPH back from exchanges and AMMs, putting them back into the distribution pool and completing the token cycle.

An automatic token burn is being considered but not finalized and hence excluded from v1 of this whitepaper.

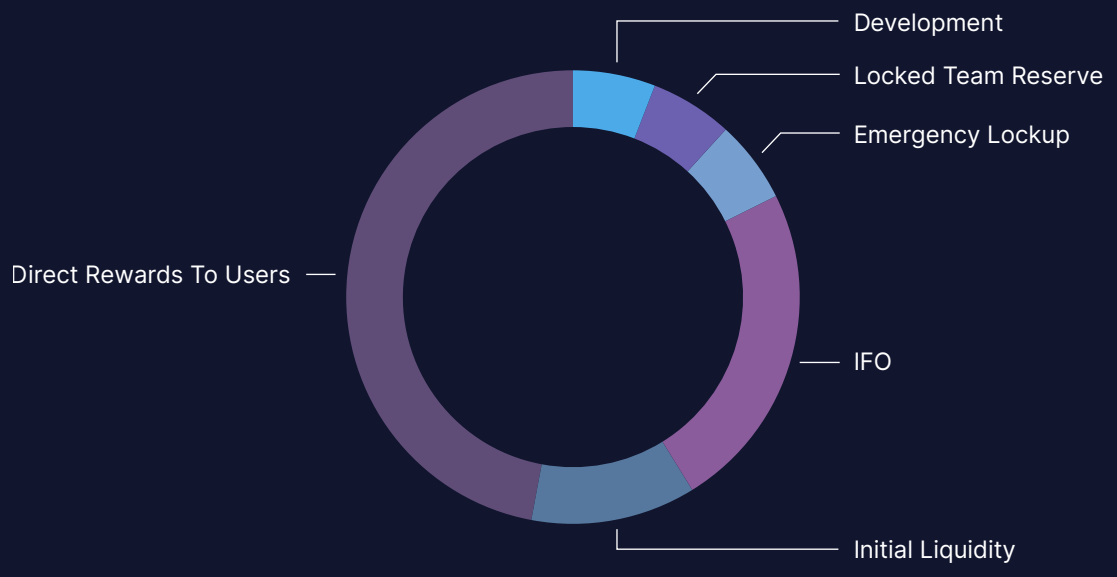
Token Overview

20.000.000 TYPH Max Supply

- **30% (6.000.000) Project Allocation**
  - 10% (2.000.000) are to be made available to the team
  - 20% (4.000.000) are to be locked away behind *immutable* linear vesting contracts and to be released over a 12 months period
    - 10% of the locked away funds is to be used for ongoing costs, potentially legal
    - 10% is to be kept as reserve and not put into circulation unless deemed necessary in emergency

- **30% (6.000.000) Initial Liquidity**
  - 20% (4.000.000) is to be used for an Initial Funds Offering and token circulation
  - 10% (2.000.000) is to be used for providing liquidity on AMMs and exchanges

- **40% (8.000.000) Project & Community**
  - The biggest allocation is solely for direct user distribution
  - To be used for direct rewards through staking, increasing the anonymity set and using the service
  - Airdrop and other reward initiatives



Roadmap

We envision the following roadmap for development of Typhoon

Q1 2021: MVP

- Initial Release to mainnet
- Token Circulation & IFO
- Build initial userbase
- Open source development

Q2 2021: Value

- Liquidity & Staking release
- Governance & Voting release
- Relayer functionality
- Finalize token & fee related proposals
- Airdrop considerations

Q3 2021: Ecosystem

- Fee-split system implementation
- Token Buyback
- Protocol integration into AMMs and partner sites

Q4 2021: ???

- Cross chain considerations
- The sky is the limit

Typhoon Network

Prior Art And Acknowledgements

- Initial idea and initial implementation - tornado.cash (https://github.com/tornadocash)
- Pedersen Hash - iden3 (https://iden3-docs.readthedocs.io/en/latest/iden3_repos/research/publications/zkproof-standards-workshop-2/pedersen-hash/pedersen.html)
- Circomlib and Solidity Verifier generation - iden3 (https://github.com/iden3/circomlib/blob/master/src/mimcsponge_gencontract.js)