charon bridge

Charon is a privacy enabled bridge. It works by deposits on the origin chain accompanied by a zero-knowledge proof[i]. The proof is then placed into a merkle tree and the root is passed to the other chain. Withdrawals can then happen to any address via a generation of proof of inclusion. It achieves privacy by breaking the link between deposits on one chain and withdrawals on another. Privacy is superior to other applications due to anonymity set being inclusive across chains, tokens, and amounts.

the vision

A token bridge that enables privacy. Anonymity is a feature that most in the crypto ecosystem claim to ascribe benefit. In practice, opt in systems get little traction and are target for regulatory action/ sanctions. By enshrining privacy at the bridge level, projects who wish to use the fungible token of a given chain will be given privacy upon bridging by default. Eventually we (as the Ethereum or crypto ecosystem) are going to have to trade a little efficiency for privacy by default. This is an easy first step. A default via an EIP or enshrined bridge system for rollups would make it way less likely for projects to get sanctioned since they're not opting in. A simple inclusion of L1 state data on L2's could make this a completely trustless mechanism (no oracle required) and the bridge would have no worse security guarantees than current bridges[1]. A standard bridge service between rollups would also afford standardization and censorship resistant benefits as many rollups currently whitelist their native bridging service and third party (multisig) bridges have arisen to fill the gap.

design

The structure used by charon is similar to a normal bridge, except deposits to the origin chain (e.g. Ethereum) are accompanied by a zk proof. Included in the proof are: destination chain, token address (of bridged token), amount being bridged, and a nullifier (or secret). The proof is added to a merkle tree on the origin chain. At some interval, the merkle root is passed to destination chains (e.g. Optimism, Polygon, and Arbitrum). Then, on a given destination chain, any address can mint tokens out of the charon bridge contract if they provide a valid proof that they deposited a specificized amount of a token with this chain as the destination. Note that any action done can be part of a batched transaction and relayed to the charon system.

the oracle

The charon bridge necessitates a way to get information about connected chains. Either a native oracle (e.g. the polygon data bridge[ii]) or another decentralized oracle structure (e.g. tellor[iii]) can be used to pass deposit information (e.g. the merke tree of deposits) from one chain to another. The oracle mechanism is a key point in security as the oracle alone can deposit valid roots to mint on the destination chain. To achieve security regarding chain finality and allowing time for

[1] https://ethresear.ch/t/cross-layer-communication-trivially-provable-and-efficient-read-access-to-the-parent-chain/15396

message passing is important. Additionally, superior privacy is achieved if parties wait to withdraw. Since the security of oracle is paramount, the charon bridge makes more sense on systems w/ native bridges such as rollups or other light client connections (e.g. IBC).

privacy features

Charon achieves privacy due to the anonymous nature of withdrawals on alternate chains and across multiple tokens. The unique aspect of charon is that deposits are made on one chain (commitment generated using destination chain logic (the destination chain is part of the secret)), the commitment is passed to many chains, and then the withdrawal happens on the destination chain. Since the commitment is passed to many chains and many tokens are deposited, assuming other deposits and commitments, anonymity is achieved since there is no way to prove which chain is the destination or which withdrawal on the destination chain corresponds to a given deposit.

circuit and cryptographic function constructions

The cryptographic functions for off-chain use are implemented in the circomlib[iv] library. Much of the circom and solidity implementations of the Merkle Trees are taken from tornado cash[v]. The Solidity implementation of the Poseidon hasher is created by Iden3[vi]. The SNARK keypair and the Solidity verifier code are generated by the authors using SnarkJS, a tool also made by Iden3.

further considerations and future versions

Full privacy of transactions and contract interactions can also be achieved via this bridge. If rather than the merkle root of all deposits, each deposit commitment is passed over, the tree can be created and maintained on the destination chain. This could allow for a structure similar to Tornado Cash Nova or even Railgun to exist where users have additional functionality built in. The downside to this approach is that gas costs for the bridge will be higher and technical complexity for the wallets simply using the bridge will be greater (shielded wallets must scan for transactions sent to them). For those reasons, the simpler version was chosen as an initial implementation.

---

[i] zero-knowledge proof - https://en.wikipedia.org/wiki/Zero-knowledge_proof
[ii] polygon data bridge - https://docs.polygon.technology/docs/pos/state-sync/state-sync/
[iii] tellor – www.tellor.io
[iv] circomlib - https://github.com/iden3/circomlib
[v] tornado nova - https://nova.tornadocash.eth.link/
[vi] iden3 - https://github.com/iden3

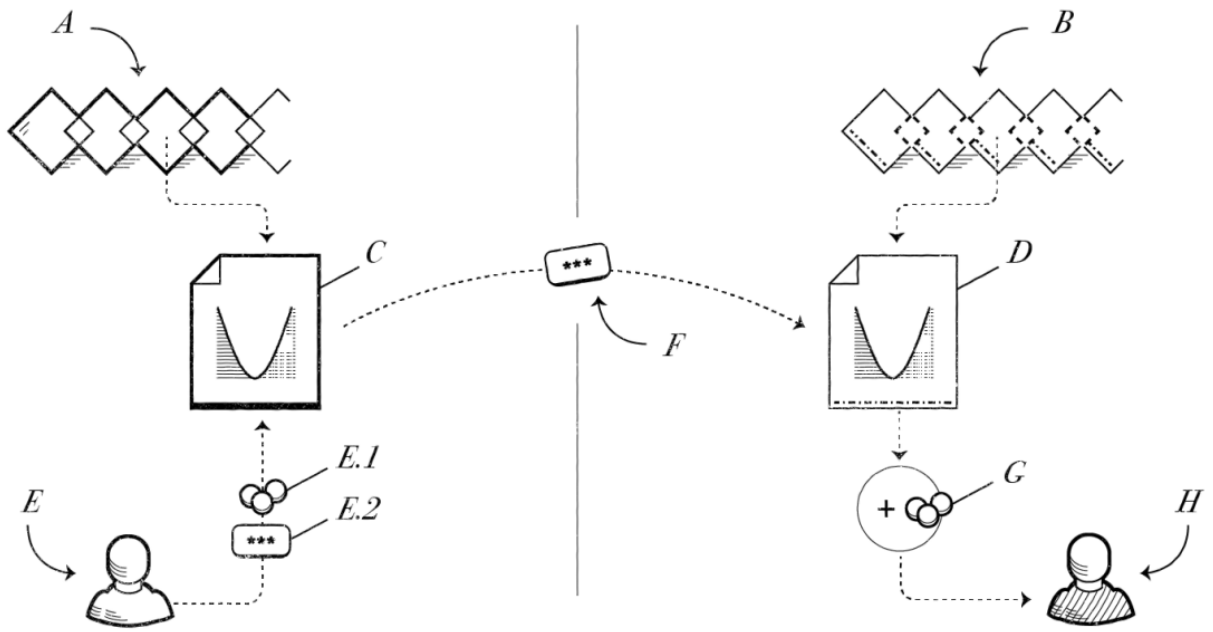

Fig. 1

a. chain 1
b. chain 2
c. charon bridge contract on chain 1
d. charon bridge contract on chain 2
e. user
    1. token deposit on chain 1
    2. commitment sent with deposit
f. oracle / native bridge passes commitment to chain 2
g. bridged tokens are minted when user presents zk proof of ownership over commitment
h. user (likely same user, but anonymous)