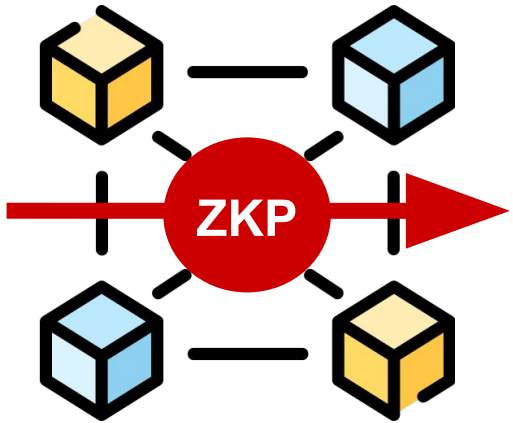
Private on-chain Line of Credit



ETH Brno 2022

Overview

Why I think this is interesting?

What have I done?

How have I did it?

Why?

Thesis: **Ethereum will become the main settlement layer for digital value. Once ZKP become standard, Enterprise Blockchain will move to public networks.**

Why?

Internet

Dedicated Connections

Blockchain

What?

Line Of Credit

A type of loan that lets you borrow money up to a pre-set limit.

What?

Features

1. **Interest rate and maximum allowance remain private** between the Client and the Financial Institution
2. Funds can be disposed 24/7 directly on chain in a trustless way
3. Withdraw any stablecoin that can be used straight away in a DeFi protocol

How?

AZTEC

Noir is a domain specific language for creating and verifying proofs

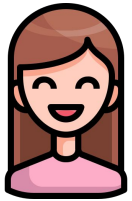
Proofs can be verified by a Smart Contract

Private On Chain Line of Credit using ZKP

1

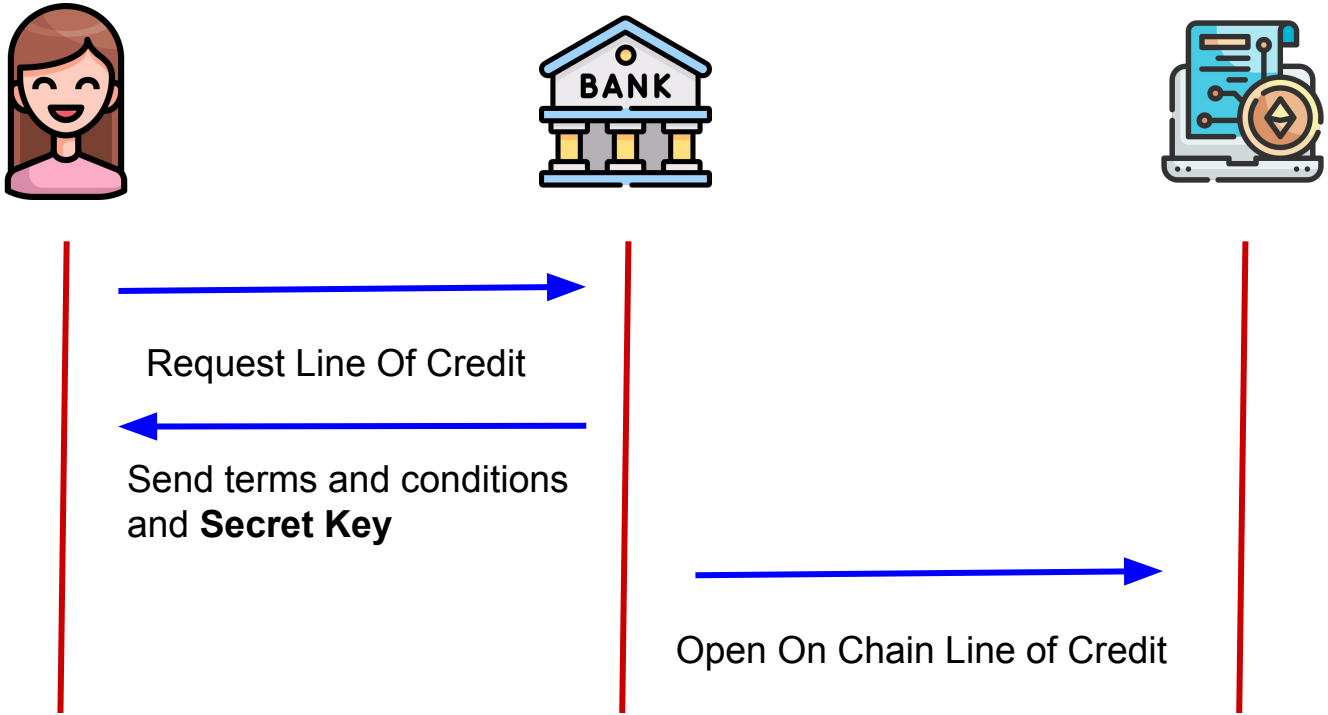
Creates Line of Credit

2

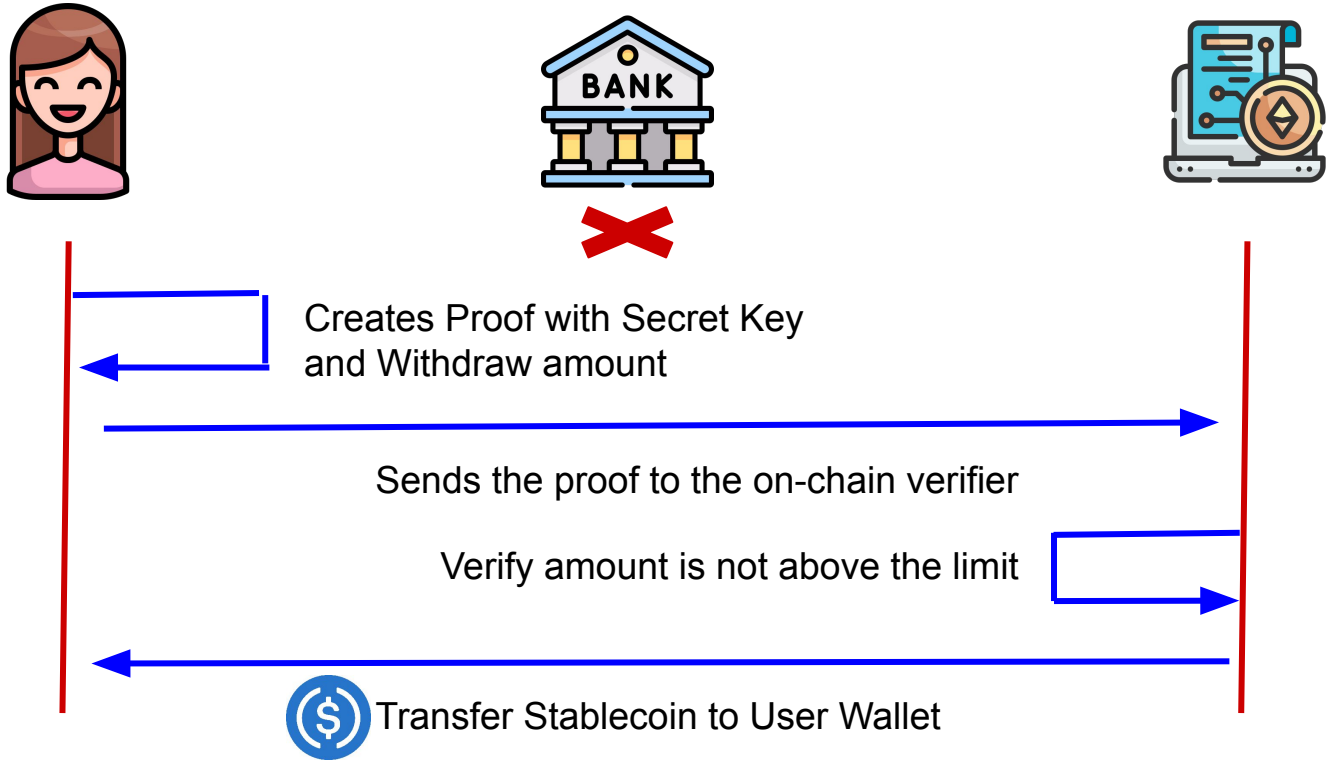


Uses Line of Credit

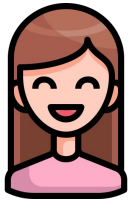
Create Line of Credit



Use Line of Credit



Final Remarks



Withdraw Stablecoin

Trustless and non-interactive way

Instant and without human interaction

Maximum amount

Interest rates

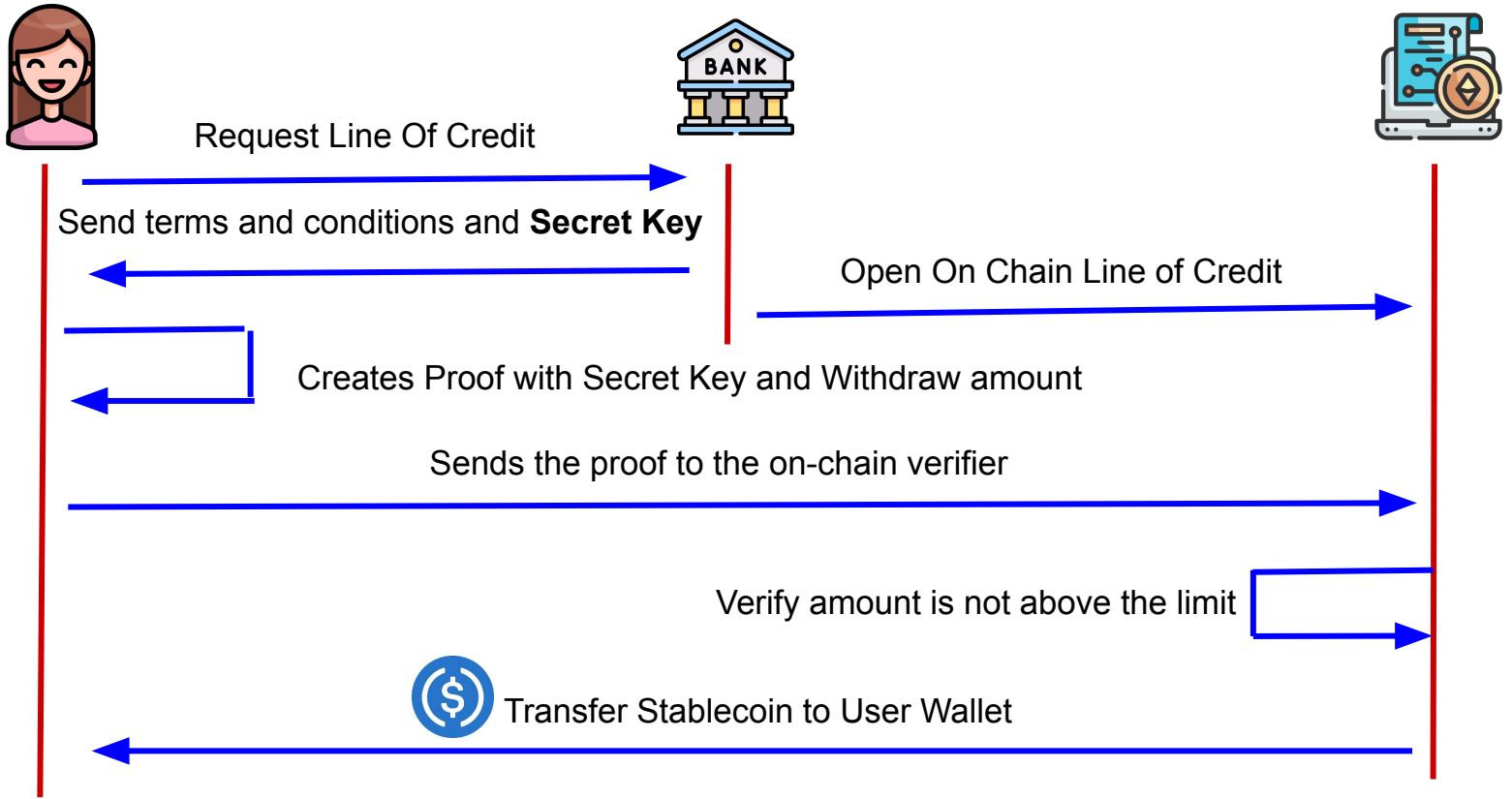
Thank you

Guillermo Barco

@gbarcoeth

gbarco.eth

Private On Chain Line of Credit using ZKP



Noir ZKP Circuit

main.nr

OnChainPrivateLineOfCredit > withdrawProofGenerator > circuits > src > main.nr

```
use dep::std;

fn main(userSecretKey: Field, withdraw: Field, totalAllowance: Field, totalAllowanceHash: pub Field) -> pub Field {
//fn main(userSecretKey: Field, withdraw: Field) -> pub Field {

    let _totalAllowanceHash = std::hash::pedersen([totalAllowance, userSecretKey]);
    constrain _totalAllowanceHash[0] == totalAllowanceHash;

    let withdrawInt = withdraw as u24;
    let totalAllowanceInt = totalAllowance as u24;
    constrain withdrawInt < totalAllowanceInt;

    let withdrawed = std::hash::pedersen([withdraw, userSecretKey]);
    withdrawed[0]
}
```

Withdraw Function

```
function withdrawWithProof(bytes calldata proof, uint256 userId) public {
    bytes32 _maximumAllowanceHash;
    uint256 _amount;
    assembly {
        _maximumAllowanceHash := calldataload(add(calldataload(0x04), 0x24))
        _amount := calldataload(add(calldataload(0x04), 0x44))
    }
    require(_maximumAllowanceHash == userId_userStatus[userId].maximumAllowanceHash, "Maximum amount not correct");
    require(turboVerifier.verify(proof) == true, "Not valid proof");
    usdFoo.transfer(msg.sender, _amount);
    emit tokensSuccessfullyCreated(_amount);
}
```