WithdrawCircuit

现在withdraw有两种方式：onchain withdraw和offchain withdraw。

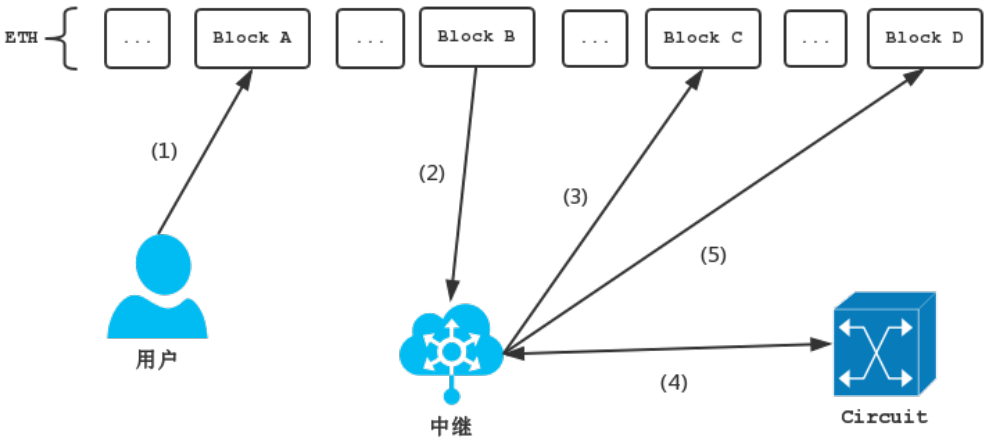
两个各有优劣：

1 onchain withdraw缺点是比较慢，要先经过以太主网确认才会到中继进行处理；优点是可以确保被处理（合约设置超过15分钟就要被优先处理）

2 offchain withdraw的优点是快，直接发到中继进行处理；缺点是中继可以选择性的不处理；

1 OnchainWithdrawCircuit

1.1 流程



1 用户发送withdraw请求到ETH主链的Exchange合约

2 中继监听Exchange合约收到用户的withdraw请求

3 中继组装withdraw请求生成withdraw的block提交，调用Exchange的commitBlock

4 中继调用电路部分生成block的prove

5 中继调用Exchange的verifyBlock

1.2 Prove Input

Prove的Input包含block的信息和每个withdraw的信息

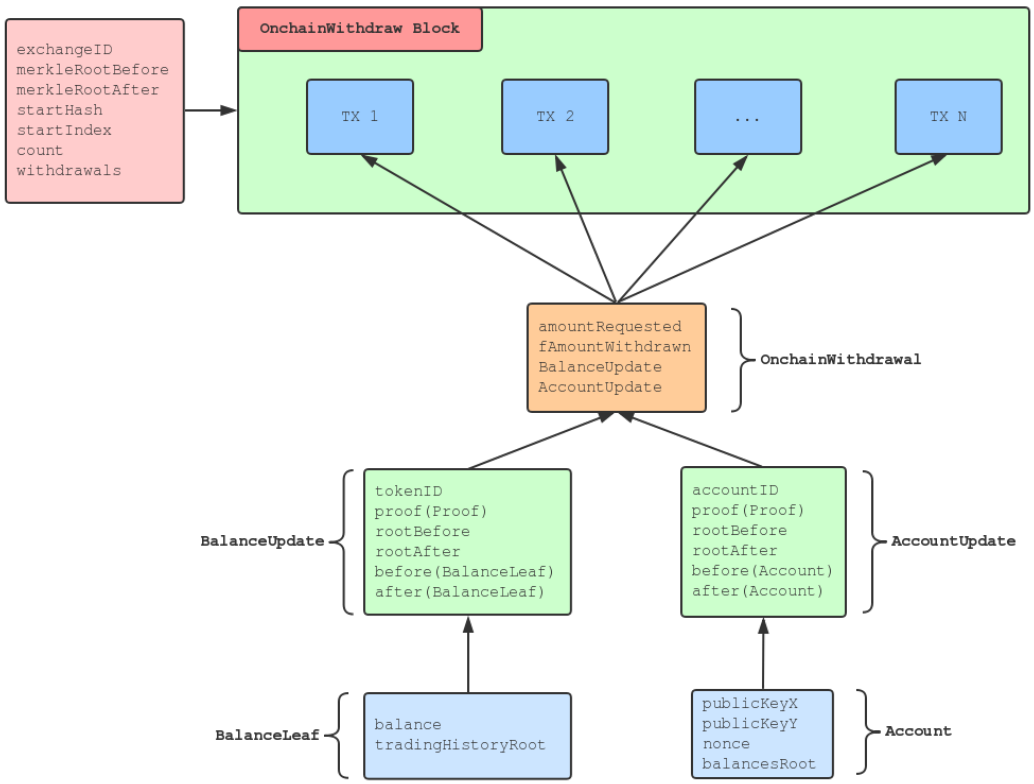
block的信息：

```
class OnchainWithdrawalBlock
{
public:

    ethsnarks::FieldT exchangeID;

    ethsnarks::FieldT merkleRootBefore;
    ethsnarks::FieldT merkleRootAfter;

    libff::bigint<libff::alt_bn128_r_limbs> startHash;

    ethsnarks::FieldT startIndex;
    ethsnarks::FieldT count;

    std::vector<Loopring::OnchainWithdrawal> withdrawals;
};
```

各个withdrawal的信息：

```
class OnchainWithdrawal
{
public:
    ethsnarks::FieldT amountRequested;
    ethsnarks::FieldT fAmountWithdrawn;
    BalanceUpdate balanceUpdate;
    AccountUpdate accountUpdate;
};

class AccountUpdate
{
public:
    ethsnarks::FieldT accountID;
    Proof proof;
    ethsnarks::FieldT rootBefore;
    ethsnarks::FieldT rootAfter;
    Account before;
    Account after;
};

class Account
{
public:
    ethsnarks::jubjub::EdwardsPoint publicKey;
    ethsnarks::FieldT nonce;
    ethsnarks::FieldT balancesRoot;
};
```

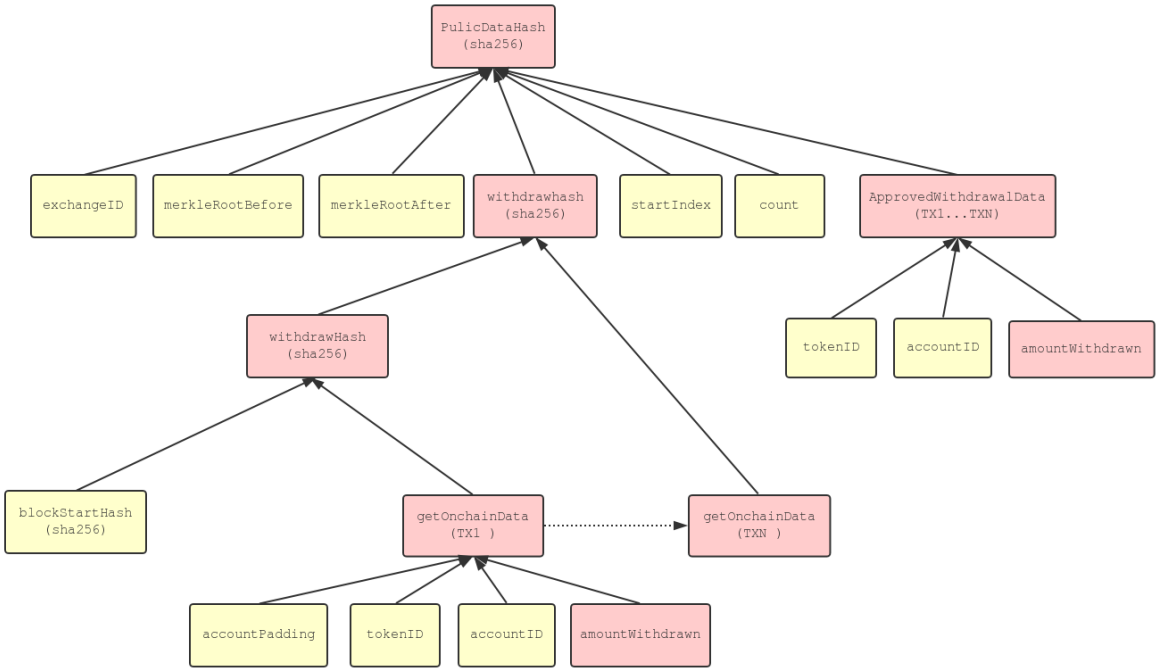
```
class BalanceUpdate
{
public:
    ethsnarks::FieldT tokenID;
    Proof proof;
    ethsnarks::FieldT rootBefore;
    ethsnarks::FieldT rootAfter;
    BalanceLeaf before;
    BalanceLeaf after;
};

class BalanceLeaf
{
public:
    ethsnarks::FieldT balance;
    ethsnarks::FieldT tradingHistoryRoot;
};
```



1.3 Circuit

1.3.1 计算publicDataHash

电路会计算一个sha256 hash，verifyBlock的时候输入hash与这个hash一致即表示verify成功。



1.3.2 OnchainWithdrawalGadget

每个withdrawal的逻辑就是验证withdraw的合法性，并且加入了shutdownmode的判断，如果是shutdownmode就要withdraw用户的所有balance，然后更新balance和account。

具体步骤如下：

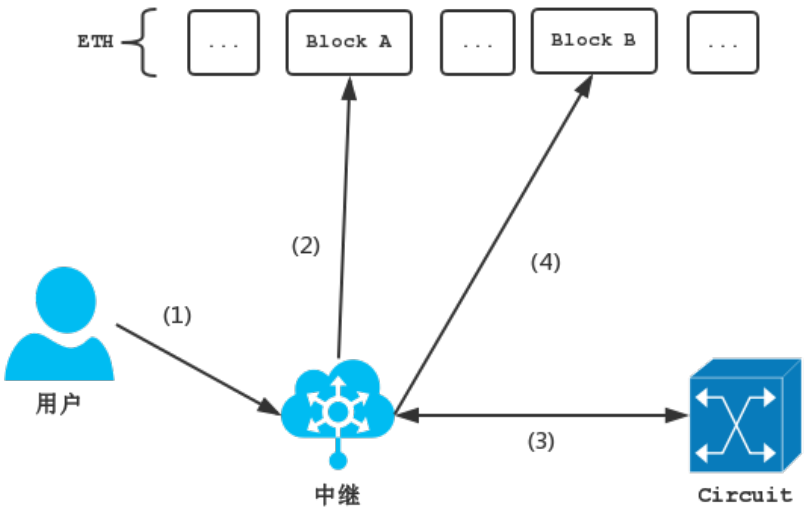
```
1 取用户amountRequested和balanceBefore的最小值算出amountToWithdrawMin
2 根据shutdownmode计算amountToWithdraw，如果是shutdownmode结果为balanceBefore,否则为amountToWithdrawMin
3 设置amountWithdrawn，值是输入的fAmountWithdrawn(toFloat(min(amountRequested, balance)), 类型Float28Encoding // {5, 23, 10};
   struct FloatEncoding
    {
        unsigned int numBitsExponent;
        unsigned int numBitsMantissa;
        unsigned int exponentBase;
    };
4 ensureAccuracyAmountWithdrawn, 确保amountToWithdraw和fAmountWithdrawn在精度范围内相等。Float28Accuracy  // {12, 10000000};
  struct Accuracy
    {
        unsigned int numerator;
        unsigned int denominator;
    };
 TODO: amountRequested和fAmountWithdrawn都是输入，有点不合理，Brecht said:
```

```
 We also don't covert to float in the circuit, this would be expensive. The
float is decoded and it's  checked that the float value is close enough to the
actual value.
 现在跟fee的处理不一样，后续会改成一致。
5 根据shutdownmode计算amountToSubtract，true则是amountToWithdraw，否则是
amountWithdrawn
6 根据shutdownmode计算tradingHistoryAfter， true则是emptyTradeHistory， 否则是
balanceBefore.tradingHistory
7 根据shutdownmode计算publicKeyXAfter， true则是zero，否则是
accountBefore.publicKeyX
8 根据shutdownmode计算publicKeyYAfter， true则是zero，否则是
accountBefore.publicKeyY
9 根据shutdownmode计算nonceAfter，true则是zero，否则是accountBefore.nonce
(可以看出onchain withdraw不改变nonce，跟deposit一样)
10 确保balanceBefore.balance = balanceAfter.balance + amountToSubtract
11 updateBalance
12 updateAccount
```

withdrawCircuit中会check最后一个用户的rootCalculatorAfter等于输入的merkleRootAfter

2 OffchainWithdrawCircuit

2.1 流程



1 用户发送withdraw请求到中继

2 中继收到用户withdraw请求。

组装withdraw请求生成withdraw的block提交到ETH主链，调用Exchange的commitBlock

3 中继调用电路部分生成block的prove

4 中继调用Exchange合约的verifyBlock

2.2 Prove Input

Prove的Input包含block的信息和每个withdraw的信息

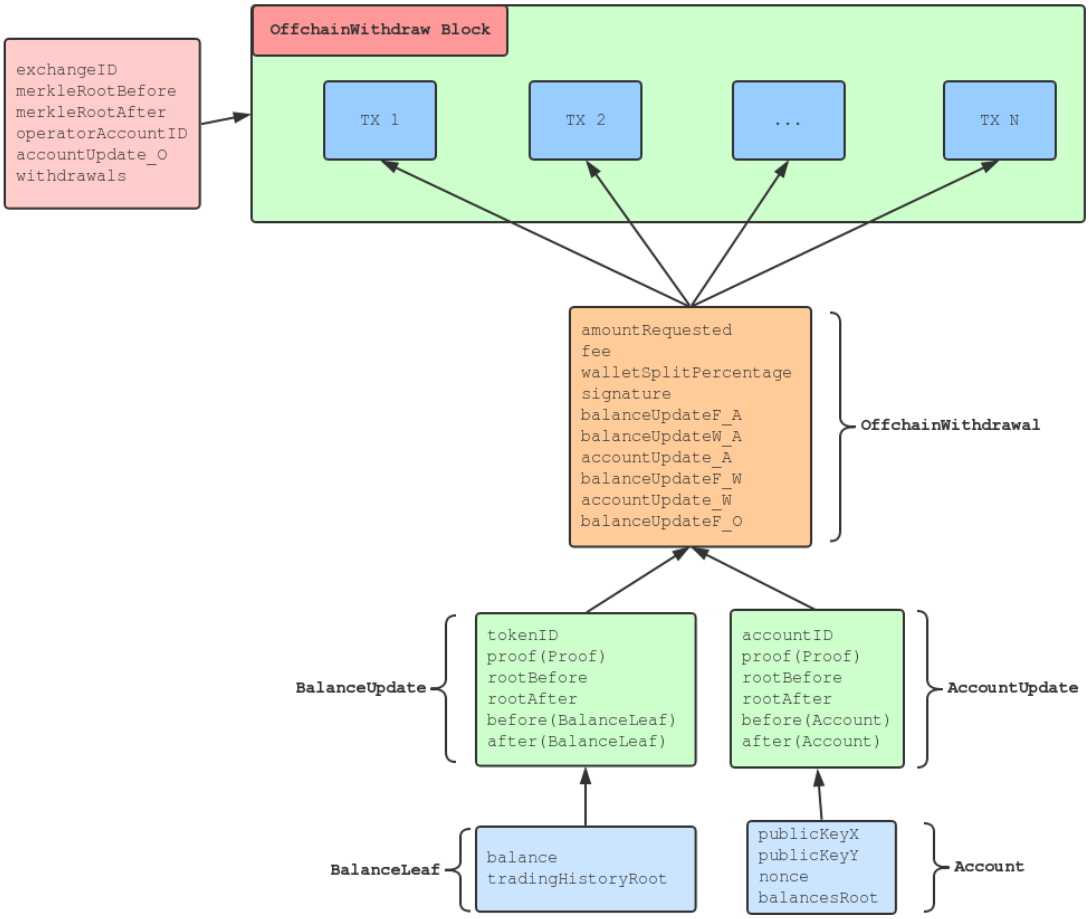
block的信息：

```
class OffchainWithdrawalBlock
{
public:

    ethsnarks::FieldT exchangeID;

    ethsnarks::FieldT merkleRootBefore;
    ethsnarks::FieldT merkleRootAfter;

    libff::bigint<libff::alt_bn128_r_limbs> startHash;

    ethsnarks::FieldT startIndex;
    ethsnarks::FieldT count;

    ethsnarks::FieldT operatorAccountID;
    AccountUpdate accountUpdate_O;

    std::vector<Loopring::OffchainWithdrawal> withdrawals;
};
```

各个withdrawal的信息：

```
class OffchainWithdrawal
{
public:
    ethsnarks::FieldT amountRequested;
    ethsnarks::FieldT fee;
    ethsnarks::FieldT walletSplitPercentage;
    Signature signature;

    BalanceUpdate balanceUpdateF_A;
    BalanceUpdate balanceUpdateW_A;
    AccountUpdate accountUpdate_A;
    BalanceUpdate balanceUpdateF_W;
    AccountUpdate accountUpdate_W;
    BalanceUpdate balanceUpdateF_O;
```

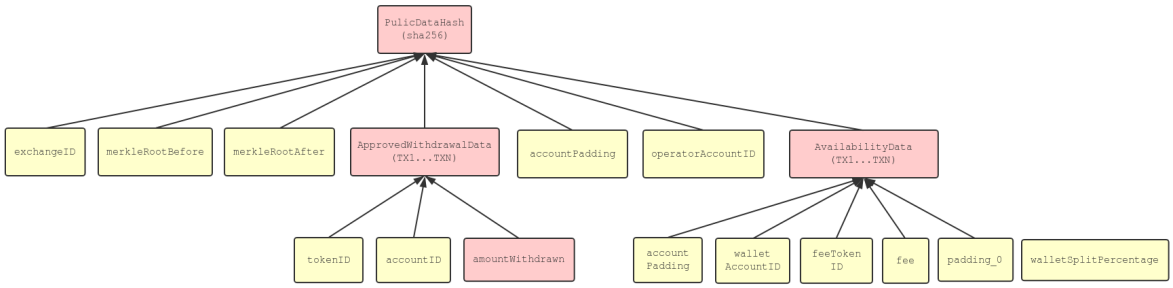
```
};
```



2.3 Circuit

2.3.1 计算publicDataHash

电路会计算一个sha256 hash，verifyBlock的时候输入hash与这个hash一致即表示verify成功。



2.3.2 OffchainWithdrawalGadget

每个withdrawal的逻辑跟onchainwithdraw的逻辑类似，但是不需要判断shutdownmode，增加了fee的操作以及验证用户的签名等逻辑。

fee需要付给wallet和operator，所以这里涉及到3个角色：用户、wallet和operator

具体步骤如下：

```
1 计算feeToWallet = fee * walletSplitPercentage
2 计算feeToOperator = fee - feeToWallet
3 计算feePaymentWallet.X = balanceFBefore.balance - feeToWallet //用户的fee
token balanceAfter_1
和feePaymentWallet.Y = balanceWalletBefore.balance + feeToWallet // wallet的
balanceAfter
4 计算feePaymentOperator.X = feePaymentWallet.X - feeToOperator // 用户fee token
balanceAfter final
和feePaymentOperator.Y = balanceF_O_before + feeToOperator // operator的
balanceAfter
5 根据用户的amountRequested计算用户的withdraw amount： amountWithdrawn
6 计算用户的balance_after = balanceBefore.balance - amountWithdrawn
7 更新user信息：
  7.1 update fee token的balance
  7.2 update withdraw token的balance
  7.3 updateAccount
8 更新wallet信息
  8.1 update wallet balance
  8.2 update wallet acount
9 更新Operator的balance
  (Operator的account是在withdrawCircuit中最后更新一次)
10 检查用户签名
  这里用的是EdDSA_Poseidon
```

withdrawCircuit中会check operator的rootCalculatorAfter等于输入的merkleRootAfter