安全审计报告

Loopring Protocol (V3)

SECBIT

Nov 15, 2019

安比（SECBIT）实验室致力于解决区块链全生态的安全问题，提供区块链全生态的安全服务。作为中国信息通信研究院区块链安全技术组成员，参与编写区块链安全白皮书和参与制定区块链安全审计规范。

安比（SECBIT）实验室智能合约审计从合约的技术实现、业务逻辑、接口规范、Gas 优化、发行风险等维度，由两组专业审计人员分别独立进行安全审计，审计过程借助于安比实验室研发的一系列形式化验证、语义分析等工具进行扫描检测，力求尽可能全面的解决所有安全问题。

1. 综述

Loopring Protocol (V3) 是用于区块链的去中心化交易协议。安比（SECBIT）实验室于 2019 年 8 月 15 日至 11 月 15 日对 Loopring V3 **智能合约**和**电路代码**进行安全审计，审计过程从**代码漏洞**，**逻辑漏洞**和**发行风险评估**三个维度对代码进行分析。审计结果表明，Loopring Protocol (V3) 并未包含致命的安全漏洞，安比（SECBIT）实验室给出了如下几点逻辑实现存疑和代码优化建议项（详见第4章节）。

风险类型	描述	风险级别	状态
实现漏洞	4.3.1 `getLRCFeeStats()` 接口定义与实现不一致	提示	已更新
实现漏洞	4.3.2 `SignatureBasedAddressWhitelist` 中 assembly code 无法实现功能	低	已更新
机制设计	4.3.3 讨论 Operator 对 maker/taker 订单选择的影响力	低	已讨论
代码规范	4.3.4 多处使用 `transferTokens()` 未检验返回值	提示	待讨论
实现漏洞	4.3.5 `transferDeposit()` 函数实现意图不明确	中	待讨论
协议设计	4.3.6 Block 中 timestamp 数据类型需考虑溢出问题	低	待讨论
电路代码	4.3.7 `TransformRingSettlementDataGadget` 电路代码存在优化空间	提示	待讨论
文档注释	4.3.8 `TakerMakerMatchingGadget` 中电路注释错误	提示	待讨论
电路代码	4.3.9 电路中 `generateKeyPair()` 调用逻辑不符合实际	低	已更新
电路代码	4.3.10 电路中 `OffchainWithdrawalBlock` 类存在冗余字段	低	待讨论

2. 项目信息

该部分描述了项目的基本信息和代码组成。

2.1 基本信息

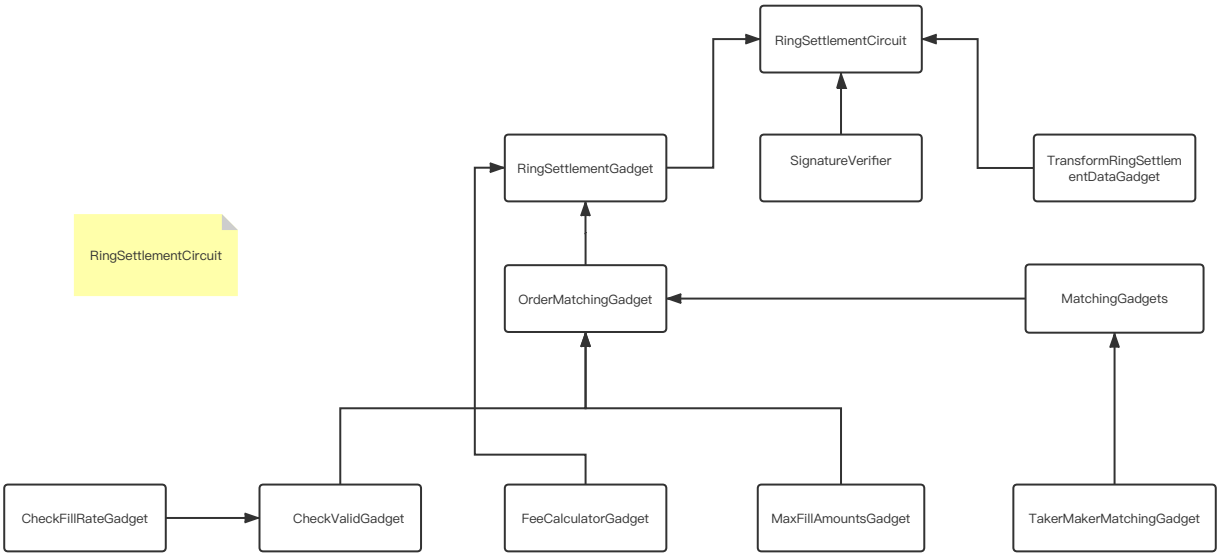
以下展示了 Loopring Protocol (V3) 的基本信息：

- **项目网站**
  - https://loopring.org/#/protocol
- **协议设计细节**
  - https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/DESIGN.md
  - https://github.com/Loopring/whitepaper
- **合约代码**
  - https://github.com/Loopring/protocols/tree/3.0-beta3/packages/loopring_v3
  - Commit `10100aa616223439516c48f2c76ef386e8f996ff`
- **电路代码**
  - https://github.com/Loopring/protocol3-circuits/tree/3.0-beta3
  - Commit `18d853d67aed649aeb2a0487870c6a37773d0d64`

2.2 电路列表

以下展示了 Loopring Protocol (V3) 项目包含的主要电路列表：

电路名称	描述
DepositCircuit	链上充值
OffchainWithdrawalCircuit	链下提现
OnchainWithdrawalCircuit	链上提现
OrderCancellationCircuit	订单取消
RingSettlementCircuit	订单环路结算



（电路 RingSettlementCircuit 内部结构）

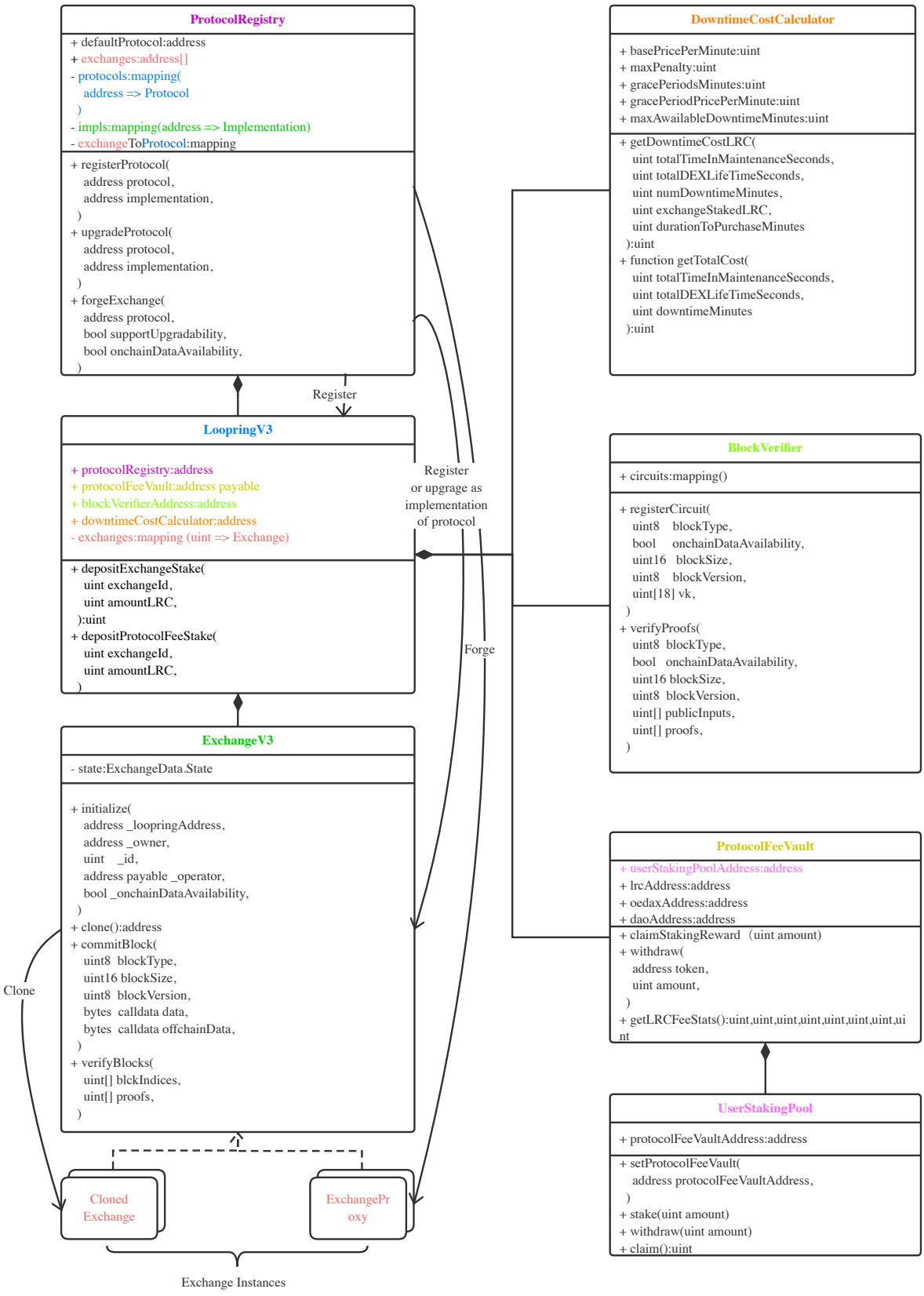
每条电路对应一类特定的业务逻辑，在链下完成计算和 zkSNARKs 证明的生成，数据和证明提交至合约进行验证。如上图所示为核心电路 `RingSettlementCircuit` 的内部结构。更多详细电路分析图参见附录。

2.3 合约列表

以下展示了 Loopring Protocol (V3) 项目包含的主要合约列表：

合约名称	描述
ProtocolRegistry	协议注册和管理入口
LoopringV3	Loopring 协议合约
ExchangeV3	Exchange 合约
ExchangeProxy	利用 Proxy 机制为 Exchange 提供可升级性
BlockVerifier	Block 验证器合约
ProtocolFeeVault	协议费保管合约
UserStakingPool	处理用户 Staking 相关逻辑
DowntimeCostCalculator	计算和管理下线时间花费
SignatureBasedAddressWhitelist	基于签名的用户白名单合约

LzDecompressor Lz 数据解压合约



（Loopring V3 智能合约组织结构）

上图为 Loopring V3 核心智能合约依赖关系和调用逻辑。合约实现可认为分为 3 层：

`ProtocolRegistry -> Protocol -> Exchange`

其中 `ProtocolRegistry` 为协议注册和管理入口，用于注册和管理各版本协议实现，`Protocol` 为协议实现，用于管理当前版本对应的子交易所，而 `Exchange` 则为具体交易所实现，可以创建和运行多个实例。

3. 代码分析

该部分描述了项目代码的详细内容分析，从「角色分类」和「功能分析」这两部分来进行说明。

3.1 角色分类

Loopring V3 中主要涉及以下几种关键角色，分别是 Registry Owner、Protocol Owner、Exchange Owner、Exchange Operator、Exchange User 和 Normal User。

- Registry Owner
  - 描述

    协议注册合约管理账户
  - 功能权限
    - 注册、升级协议
    - 禁用、启用协议
    - 设置默认协议
  - 授权方式
    - `ProtocolRegistry` 合约的创建者自动成为 `owner`
    - 可通过 `transferOwnership` 完成权限转移
- Protocol Owner
  - 描述

    协议管理员
  - 功能权限
    - 更新协议设置
    - 更新协议费用设置
  - 授权方式
    - `Protocol` 合约创建者自动成为 `owner`
    - 可通过 `transferOwnership` 完成权限转移
- Exchange Owner

    - 描述

      交易所管理员

    - 功能权限
        - 注册 Token (类似于上币)
        - 开启/关闭 特定 Token 的充值功能
        - 交易所运营抵押资产的提现
        - 处理误转入交易所地址的资产
        - 设置 Exchange Operator（交易所操作员，负责提交状态）
        - 设置交易所费用
        - 设置白名单合约
        - 开启/关闭维护模式
        - 妥善处理完用户资产后可按规定关停交易所
    - 授权方式
        - `Exchange` 合约创建者自动成为 `owner`
        - 可通过 `transferOwnership` 完成权限转移
- Exchange Operator
    - 描述

      交易所操作员

    - 功能权限
        - 提交“区块” `commitBlock()`
        - 提交“区块”证明 `verifyBlocks()`
        - 作废失效“区块” `revertBlock()`
        - 提取“区块”手续费 `withdrawBlockFee()`
    - 授权方式

      由 Exchange Owner 设定
- Exchange User
    - 描述

      交易所用户

    - 功能权限
        - 账户注册与更新
        - 充值与提现
    - 授权方式

      用户自行通过合约进行注册
- Normal User
    - 描述

      普通以太坊账户

    - 功能权限

        - 执行合约允许的其他操作
    - 授权方式

      无需授权

3.2 功能分析

Loopring V3 采用了类似 zk-rollup 的设计思路。zk-rollup 使用一颗 Merkle Tree 存储账户状态。合约则保存 Merkle Root。树的子节点为 `AccountID => (pubkey, balance)` 的映射。这里把 zk-rollup 最简化成不同账户间单币种的转账。

而交易所（如 Loopring）则需要维护更多信息，如每个币种的余额和订单历史记录。于是 Loopring 使用了 Sub-Merkle Tree，分别维护 Account、Balance 和 TradeHistory，并使用 Quad-Merkle Tree 结构和 Poseidon Hash 来进一步提高性能。

Loopring V3 用一棵 Merkle Tree 来存储子交易所所有状态，而 Exchange 合约仅保存 Merkle Tree 的 Root，其余状态均在链下保存和计算。同时还利用 `calldata` 存储关键数据，在降低链上存储成本的前提下保证「数据可用性」（data-availability）。

实现的关键功能分为以下几项：

- `commitBlock()` 提交区块

  由 Exchange Operator 负责向智能合约提交“区块” Block（注意与区块链中的区块概念区分，此处为合约中的虚拟区块）。总共有 5 种类型的 Block，与前面电路种提到的 5 种业务逻辑一一对应。一个 Block 将同一类型的业务（Request）批量打包。通过把相同类型的 Request 打包进入同一个区块，来进行批量验证，提高效率。

- `generateProof()` 生成证明

  Exchange Operator 在链下负责维护整个交易所的状态（Merkle Tree）。用户的充值、提现、订单成交或取消都对 Merkle Tree 进行更新。电路代码中有对状态更新正确性的校验，即电路中的约束条件，用以限制 Operator 必须按照用户真实操作进行更新。zkSNARKs 电路代码借助 `EthSnarks` 库实现。每条电路都需要进行预先的 `trusted setup`，生成一对 `proving key` 和 `verification key`。Operator 需要使用相应的 `proving key` 完成每次的证明。

- `verifyBlcoks()` 验证区块

  Exchange Operator 链下为每个 Block 生成 zkSNARKs 证明，再提交合约，利用相应的 `verification key` 进行验证。生成证明时间相对较长，因此这里将 Block 的提交和验证进行分离。Operator 可以乱序进行 proof 的提交（证明），一个 Block 会有 `COMMITTED` `VERIFIED` `FINALIZED` 三种状态。验证通过的 Block 状态由 `COMMITTED` 变为 `VERIFIED`。如果该区块之前所有区块均已被验证，则认为这些区块为 `FINALIZED`。在子交易所开启了「数据可用性」（data-availability）的情况下，Operator 还必须在 `commitBlock()` 时提交更多的详细

数据，才能成功验证区块。

- `deposit()` 充值

  用户调用合约进行充值。Operator 必须在一定时间内处理请求，即将用户的充值信息打包进入 Block，提交合约并完成验证。

- `withdraw()` 提现

  用户有**多种方式**进行提现，最常用的两种分别称为 OnchainWithdrawal 和 OffchainWithdrawal。Offchain 为用户向 Operator 发起请求，用户无需调用合约，更轻便，但存在 Operator 故意不处理的可能性。而 Onchain 为用户手动在合约发起提现请求，Operator 必须在一定时间内处理（与充值类似）。

  特别地，Loopring V3 还考虑了 Operator 在极端情况下（放弃押金，停止维护 Exchange 合约，交易所进入提现模式）用户资产的处理。在提现模式下，用户仅需提供资产的 Merkle Proof 就可以正常提走属于自己的资产。

4. 审计详情

该部分描述合约审计流程和详细结果，并对发现的问题（代码漏洞，代码规范和逻辑漏洞），合约发行的风险点和附加提示项进行详细的说明。

4.1 审计过程

本次审计工作，严格按照安比（SECBIT）实验室审计流程规范执行，从代码漏洞，逻辑问题以及合约发行风险三个维度进行全面分析。审计流程大致分为四个步骤:

- 各审计小组对代码进行逐行分析，根据审计内容要求进行审计
- 各审计小组对漏洞和风险进行评估
- 审计小组之间交换审计结果，并对审计结果进行逐一审查和确认
- 审计小组配合审计负责人生成审计报告

4.2 审计结果

本次审计首先经过安比（SECBIT）实验室推出的分析工具 adelaide、 sf-checker 和 badmsg.sender（内部版本）扫描，再利用开源安全分析工具 Mythril、Slither、SmartCheck 以及 Securify 检查，检查结果由审计小组成员详细确认。审计小组成员对合约源码和电路代码进行逐行检查、评估，汇总审计结果。审计内容总结为如下 21 大项。

编号	分类	结果

1	合约各功能能够正常执行	通过
2	合约代码不存在明显的漏洞（如整数溢出）	通过
3	能够通过编译器的编译并且编译器没有任何警告输出	通过
4	合约代码能够通过常见检测工具检测，并无明显漏洞	通过
5	不存在明显的 Gas 损耗	通过
6	符合 EIP20 标准规范	通过
7	底层调用（call，delegatecall，callcode）或内联汇编的操作不存在安全隐患	通过
8	代码中不包含已过期或被废弃的用法	通过
9	代码实现清晰明确，函数可见性定义明确，变量数据类型定义明确，合约版本号明确	通过
10	不存在冗余代码	通过
11	不存在受时间和外部网络环境影响的隐患	通过
12	业务逻辑实现清晰明确	通过
13	代码实现逻辑与注释，项目白皮书等资料保持一致	通过
14	代码不存在设计意图中未提及的逻辑	通过
15	业务逻辑实现不存在疑义	通过
16	不存在危及项目方利益的明确风险	通过
17	不存在危及相关机构如交易所，钱包，DAPP 方利益的明确风险	通过
18	不存在危及普通持币用户利益的明确风险	通过
19	不存在修改他人账户余额的特权	通过
20	不存在铸币权限	通过
21	多管理角色下，管理权限划分明确，各权限优先级划分明确	通过

4.3 问题列表

4.3.1 `getLRCFeeStats()` 接口定义与实现不一致

风险类型	风险级别	影响点	状态
文档注释	提示	文档和实现不一致	已更新

问题描述

`IProtocolFeeVault` 中 `getLRCFeeStats()` 的文档注释与代码实现不符，疑似顺序颠倒。

```
    /// @return accumulatedDAOFund The accumulated amount of LRC to
burn.
    /// @return accumulatedBurn The accumulated amount of LRC as
developer pool.
```

修改建议

修正注释即可。

状态

新版本中已自行更新。

4.3.2 `SignatureBasedAddressWhitelist` 中 assembly code 无法实现功能

风险类型	风险级别	影响点	状态
代码实现	低	功能失效	已修复

问题描述

```
    assembly {
        t := mload(add(permission, 8)) // first 8 bytes as time in
second since epoch
        ...
    }
```

注释中想取 `first 8 bytes`，上述代码实际取值包含了 `permission` 的 length 信息，因此 `t` 变量提取完全错误。此外该合约缺乏测试用例。

修改建议

对提取后的数据进行截断处理，补充测试用例。

状态

新版本中已自行更新。

4.3.3 讨论 Operator 对 maker/taker 订单选择的影响力

风险类型	风险级别	影响点	状态
机制设计	低	在有限范围内影响成交价格	已讨论

问题描述

设计文档中提到 Operator 决定订单是 maker 还是 taker。

> *The operator chooses which order is the maker and which order is the taker. The first order in the ring is always the taker order, the second order is always the maker order. We always use the rate of the second order for the trade.*

考虑此处是否存在操纵空间?

以一个订单环路匹配为例，如果两个订单都是 `buy order`。注：`buy order` 选项用于帮助 taker 获得价差。

Case A	Taker (A)	Maker (B)
tokenS	WETH	GTO
tokenB	GTO	WETH
amountS	101	200
amountB	100	200
isBuy	true	true
fill.S	101 -> 100 (spread = 1)	100
fill.B	100	100

target price (WETH/GTO)	1.01	1
settle price (WETH/GTO)	1	1
balance tokenS	1	100
balance tokenB	100	100

在 `Case A` 例子中，Operator 最终以 `1` 的价格撮合订单，A 作为 Taker 拿到了价差。

如果 Operator 更换 taker 和 maker 的身份，则有下表。

Case B	**Taker (B)**	**Maker (A)**
tokenS	GTO	WETH
tokenB	WETH	GTO
amountS	200	101
amountB	200	100
isBuy	true	true
fill.S	101 -> 100 (spread = 1)	101
fill.B	101	100
target price (WETH/GTO)	1	1.01
settle price (WETH/GTO)	1.01	1.01
balance tokenS	100	0
balance tokenB	101	100

在 `Case B` 中，Operator 以 `1.01` 的价格撮合订单，B 作为 Taker 拿到了价差。

因此，Operator 存在一定空间可选择哪个用户获得更好的价格。

状态

Brecht Devos 在 issue 中的回复。

Operator 对于 order-matching 有决定权，由于用户最终成交价格没有比其预期差，Operator 在有限范围内影响价格是可以接受的。

4.3.4 多处使用 `transferTokens()` 未检验返回值

风险类型	风险级别	影响点	状态
代码规范	信息	不符合安全开发规范	待讨论

问题描述

withdrawFromMerkleTreeFor() 和 withdrawFromDepositRequest() 函数中使用 transferTokens() 未校验返回值。

```
    // Transfer the tokens
    transferTokens(
        S,
        _deposit.accountID,
        _deposit.tokenID,
        amount,
        false
    );
```

上面两处该函数使用时传入 `allowFailure = false`，故异常情况会提前 revert，因而此处不校验返回值无风险。

修改建议

增加校验或注释说明。

状态

待讨论。

4.3.5 `transferDeposit()` 函数实现意图不明确

风险类型	风险级别	影响点	状态
实现漏洞	中	易发生歧义	已更新

问题描述

transferDeposit() 函数实现中，对 ETH 取的是 `msg.sender` 转入的资产，但对 token 取的则是 `accountOwner` 的资产。

```
    function transferDeposit(
        address accountOwner,
        ...
        )
        private
    {
        ...
        require(msg.value >= totalRequiredETH, "INSUFFICIENT_FEE");
        uint feeSurplus = msg.value.sub(totalRequiredETH);
        msg.sender.transferETH(feeSurplus, gasleft());
        ...
                tokenAddress.safeTransferFrom(
                    accountOwner,
                    address(this),
                    amount
                )
        ...
    }
```

在充值场景下，从两处取资产似乎并不合理。当前 `depositTo()` 函数实现的功能为：他人为 `accountOwner` 交 ETH 手续费，并用 `accountOwner` 自身的资产进行充值。而原本意图可能为他人完全使用自己的资产为 `accountOwner` 进行充值。实现和意图之间可能存在出入。

修改建议

`transferDeposit()` 函数直接去除无意义的 `accountOwner` 参数，`safeTransferFrom()` 第一个参数传入 `msg.sender`，与前面保持一致。

状态

新版本中已自行更新，方案为使用 `msg.sender` 作为参数传入。建议直接去掉该参数。待进一步讨论。

4.3.6 Block 中 timestamp 数据类型需考虑溢出问题

风险类型	风险级别	影响点	状态
协议设计	低	时间无法表示，影响协议稳定性	待讨论

问题描述

`Block` 数据结构中使用 `uint32` 来表示 timestamp，或存在溢出风险。2106 年后协议中的时间或将无法表示。

```
    struct Block
    {
        // The time the block was created.
        uint32 timestamp;
    }
```

Bitcoin 中目前使用 `uint32` 类型，而 Ethereum 使用 `uint256`。在溢出时间点到临前应该有足够长的时间进行协议升级。

修改建议

知悉此问题，讨论将 `uint32` 换成 `uint256` 的可能性和必要性。

状态

待讨论。

4.3.7 `TransformRingSettlementDataGadget` 电路代码存在优化空间

风险类型	风险级别	影响点	状态
电路代码	提示	电路冗余代码	待讨论

问题描述

`RingSettlementCircuit.h` 中 `TransformRingSettlementDataGadget` 的 `generate_r1cs_constraints` 函数能够进一步优化为以下代码。

```
struct Range
{
    unsigned int offset;
    unsigned int length;
};

std::vector<Range> ranges;
ranges.push_back({{0, 40}});                    // orderA.orderID +
orderB.orderID
ranges.push_back({{40, 40}});                   // orderA.accountID +
orderB.accountID
ranges.push_back({{80, 8}, {120, 8}});          // orderA.tokenS +
orderB.tokenS
ranges.push_back({{88, 24},{128, 24}});         // orderA.fillS +
orderB.fillS
ranges.push_back({{112, 8}});                   // orderA.data
ranges.push_back({{152, 8}});                   // orderB.data
for(const Range& subRange : ranges)
{
```

```
    for (unsigned int i = 0; i < numRings; i++)
    {
        transformedData.add(subArray(data, i * ringSize +
subRange.offset, subRange.length));
    }
}
```

电路中无必要做数据压缩操作，`compressedData` 变量及相关处理已无用，原来代码存在较多冗余操作。

修改建议

考虑将 `compressedData` 相关代码移除。

状态

待讨论。

4.3.8 `TakerMakerMatchingGadget` 中电路注释错误

风险类型	风险级别	影响点	状态
文档注释	信息	注释不准确	待讨论

问题描述

```
takerFillB_lt_makerFillS(pb, takerFill.B, makerFill.S,
NUM_BITS_AMOUNT, FMT(prefix, ".takerFill.B < makerFill.B")),
```

修改建议

改为 `.takerFill.B < makerFill.S`。

状态

待讨论。

4.3.9 电路中 `generateKeyPair()` 调用逻辑不符合实际

风险类型	风险级别	影响点	状态
电路代码	低	不规范的流程	已更新

问题描述

电路 main.cpp 中 generateKeyPair() 调用逻辑不符合生产需求。

```
    if (mode == Mode::CreateKeys || mode == Mode::Prove)
    {
        if (!generateKeyPair(pb, baseFilename))
        {
            std::cerr << "Failed to generate keys!" << std::endl;
            return 1;
        }
    }
```

当 Mode 为 Prove 和 CreateKeys 时，电路代码都尝试进行 generateKeyPair()。而实际生产环境下，generateKeyPair() 需要 trusted setup，需要多人参与并且在前期只需产生一次。目前发现 KeyPair 不存在就进行重新生成的逻辑，不符合实际。

修改建议

将两个动作分离，且当 pk 和 vk 不存在时，不允许执行 Prove 操作。

状态

新版本中已自行更新。

4.3.10 电路中 `OffchainWithdrawalBlock` 类存在冗余字段

风险类型	风险级别	影响点	状态
电路代码	低	冗余字段	待讨论

问题描述

Data.h 文件 OffchainWithdrawalBlock 类中，startIndex 和 count 字段未使用。

```
class OffchainWithdrawalBlock
{
public:
    ethsnarks::FieldT exchangeID;
    ethsnarks::FieldT merkleRootBefore;
    ethsnarks::FieldT merkleRootAfter;
    libff::bigint<libff::alt_bn128_r_limbs> startHash;
    ethsnarks::FieldT startIndex; // unused
    ethsnarks::FieldT count; // unused
    ethsnarks::FieldT operatorAccountID;
    AccountUpdate accountUpdate_O;
    std::vector<Loopring::OffchainWithdrawal> withdrawals;
};
```

修改建议

去除未使用的多余字段。

状态

待讨论。

4.4 风险提示

4.4.1 需考虑极端情况下的异常处理

风险类型	风险级别	影响点	状态
参数设置	中	协议稳定性	待讨论

问题描述

目前 `ExchangeData.sol` 中大量时间参数是一次性硬编码的。合约对链上请求的处理和订单时间的有效性有着严格的时限限制，需考虑极端情况下协议是否依然能稳定运行。而这些时间参数大多又涉及对 Operator 权限的限制，需要谨慎选择取值。

如 `MAX_PROOF_GENERATION_TIME_IN_SECONDS`、`MAX_AGE_REQUEST_UNTIL_WITHDRAW_MODE`、`TIMESTAMP_HALF_WINDOW_SIZE_IN_SECONDS` 等关键参数需要尤其重视。

建议

- 需重点考虑或模拟极端情况，根据如交易量过大、以太坊网络过于拥堵、Operator 后台程序不稳定等场景选取时间参数极限值
- 后台程序在涉及区块链网络拥堵、交易发送和监控、分叉处理、抗 DOS 攻击等异常处理方面需要足够健壮，需充分测试后上线

状态

待讨论。

5. 结论

Loopring Protocol (V3) 充分利用 zkSNARKs 技术完整地设计并实现了高性能的去中心化交易协议。安比（SECBIT）实验室在对 Loopring Protocol (V3) 合约和电路代码进行审计分析后，发现了部分可优化项，并提出了对应的修复及优化建议，上文均已给出具体的分析说明。尤其值得一提的是，安比（SECBIT）实验室认为，Loopring Protocol (V3) 项目代码质量较高，代码结构清晰、函数命名规范、注释完整，并且拥有完善的测试用例。Loopring 首次实现了复杂应用的电路代码，使得链上计算得以显著压缩。在零知识证明技术的落地和 Layer-2 扩容方案的研究上成果丰富。Loopring 开发团队对交易性能和去中心化程度的追求、对用户资产安全性所做的努力，是令人钦佩的。

免责声明

SECBIT 智能合约安全审计从合约代码质量、合约逻辑设计和合约发行风险等方面对合约的正确性、安全性、可执行性进行审计，但不做任何和代码的适用性、商业模式和管理制度的适用性及其他与合约适用性相关的承诺。本报告为技术信息文件，不作为投资指导，也不为代币交易背书。

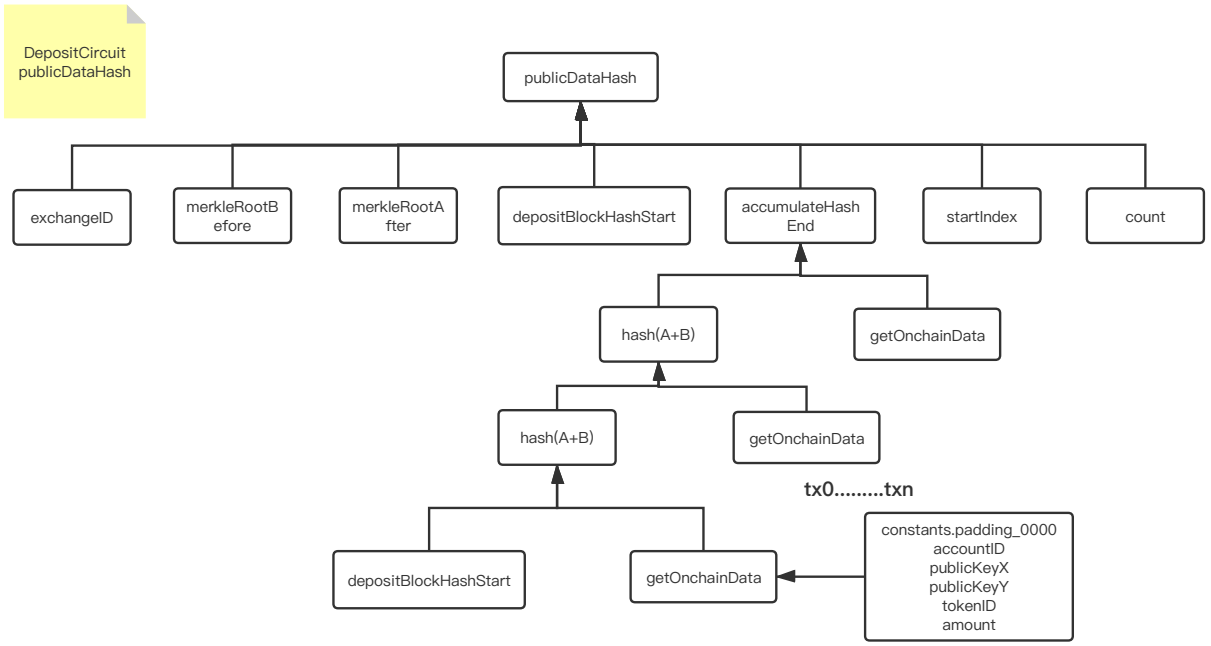
附录

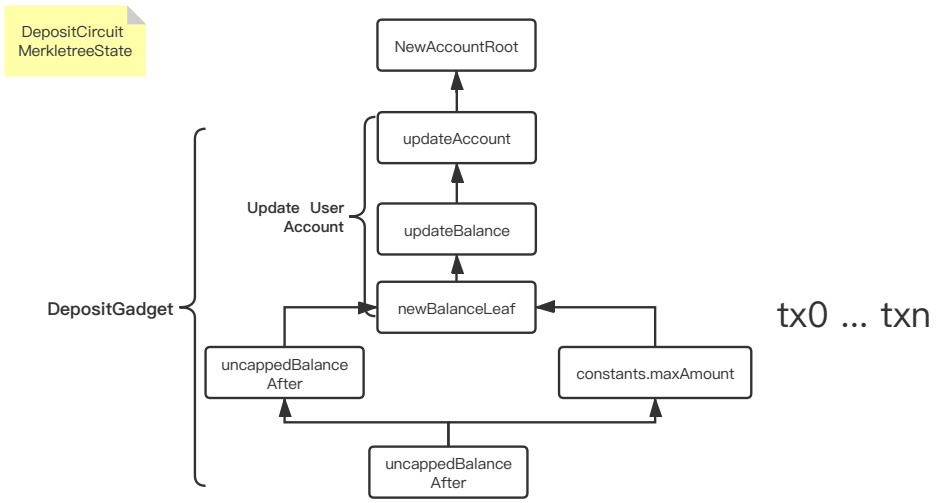
漏洞风险级别介绍

风险级别	风险描述
高	可以严重损害合约完整性的缺陷，能够允许攻击者盗取以太币及Token，或者把以太币锁死在合约里等缺陷。
中	在一定限制条件下能够损害合约安全的缺陷，造成某些参与方利益损失的缺陷。
低	并未对合约安全造成实质损害的缺陷。
提示	不会带来直接的风险，但与合约安全实践或合约合理性建议有关的信息。

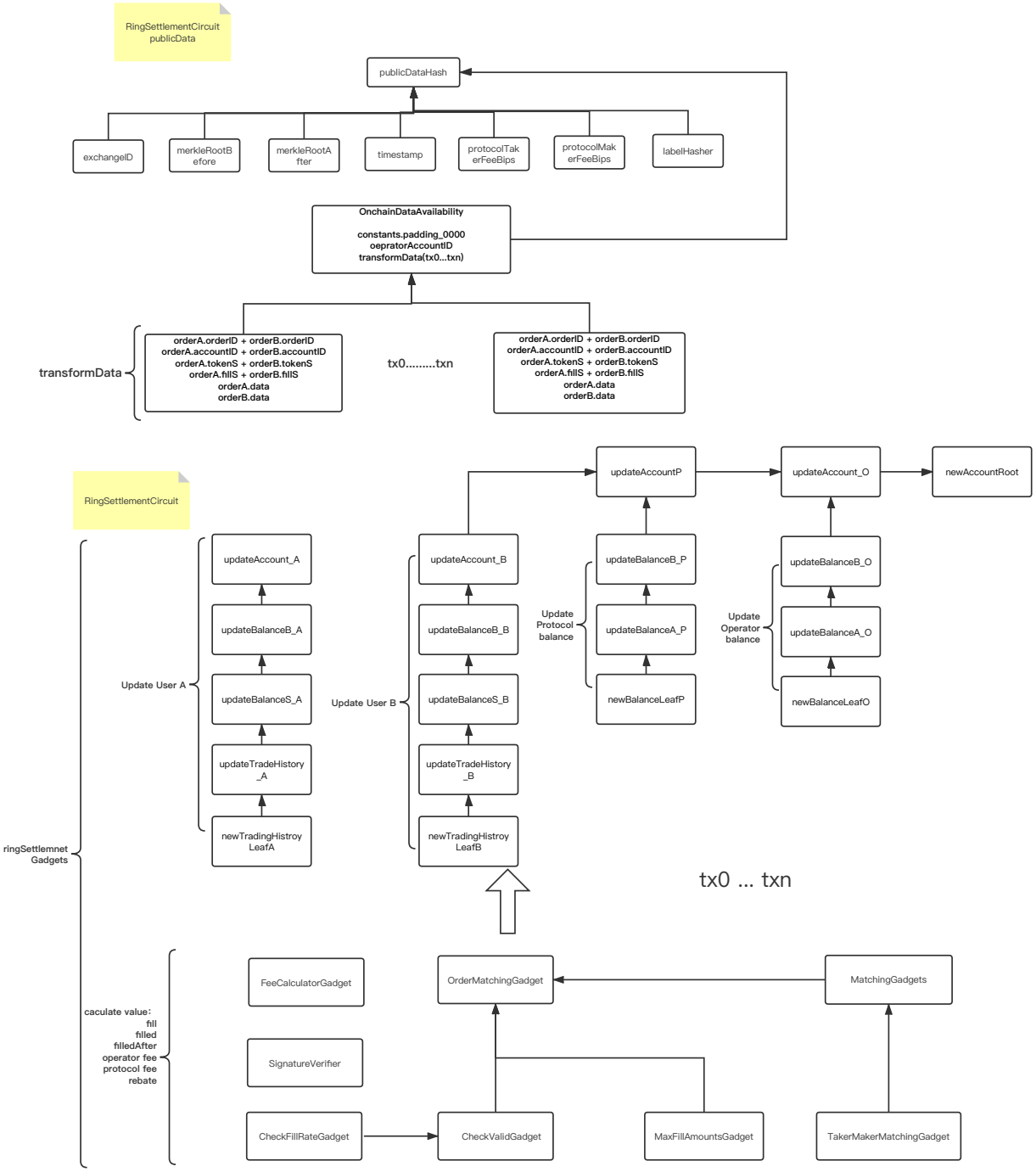
电路实现分析图

DepositCircuit

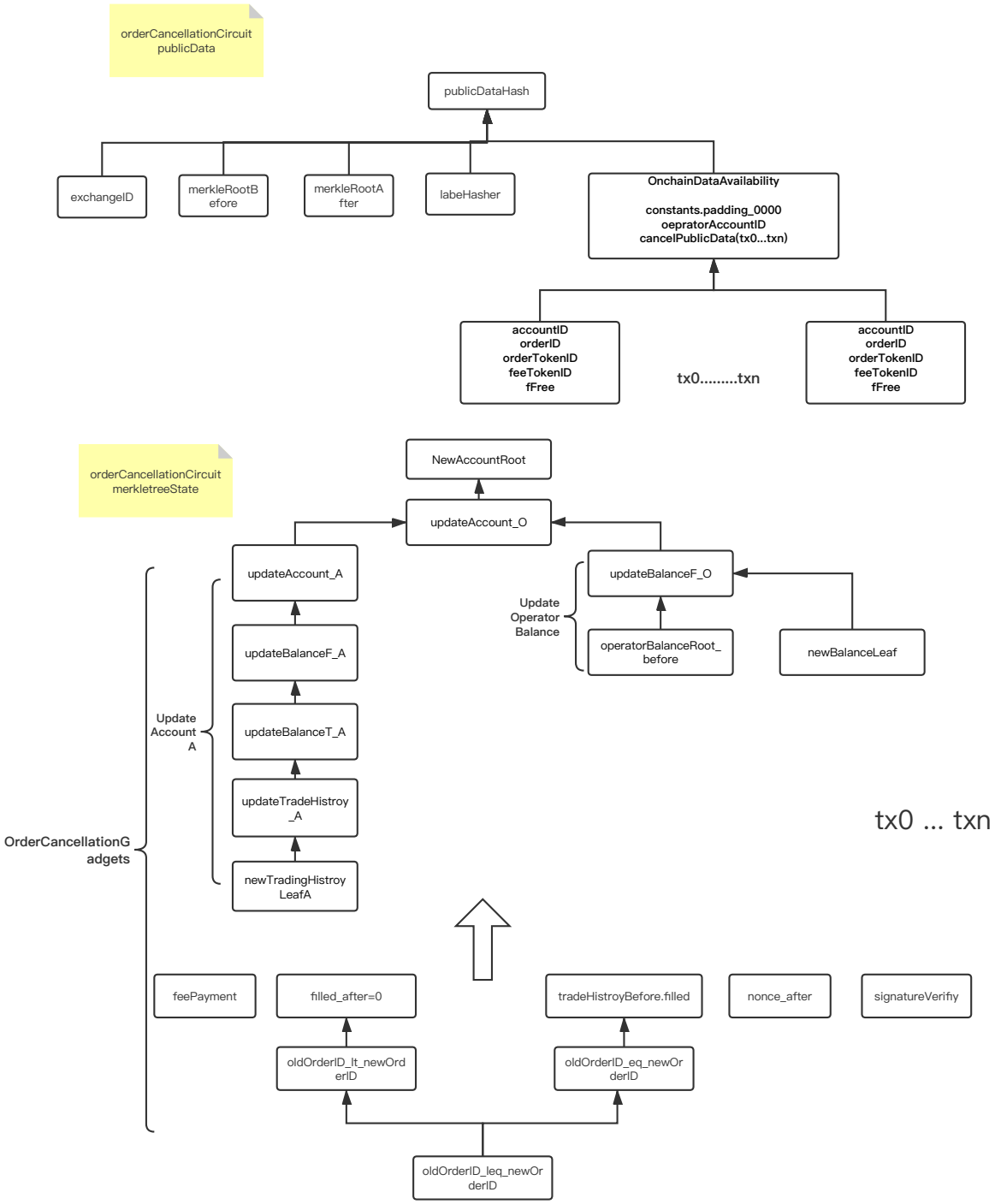




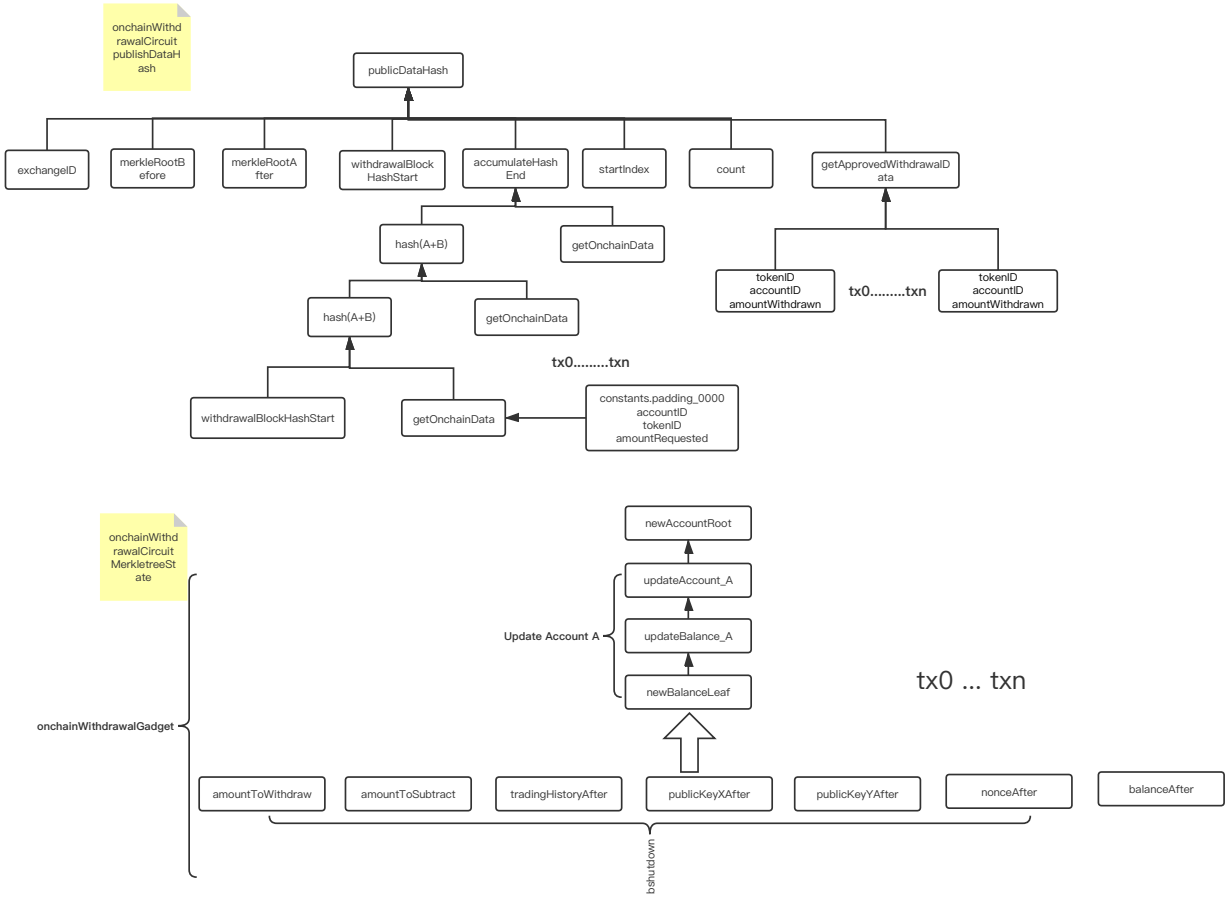
RingsettlementCircuit



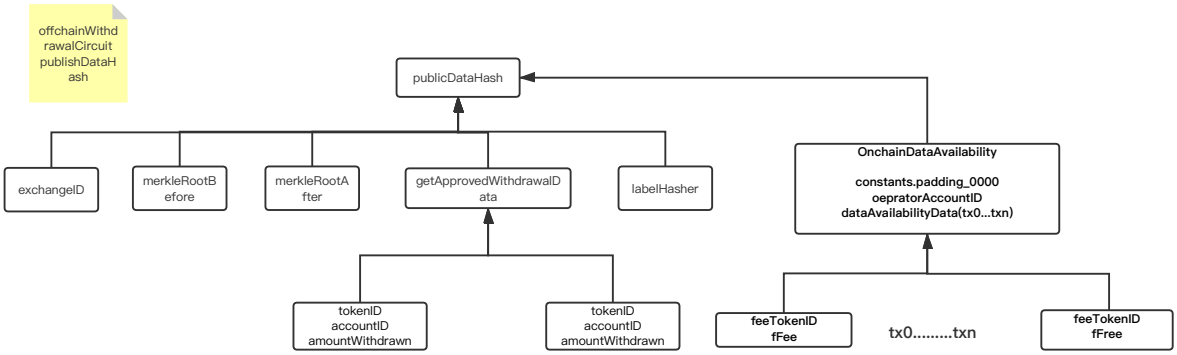
OrderCancellationCircuit

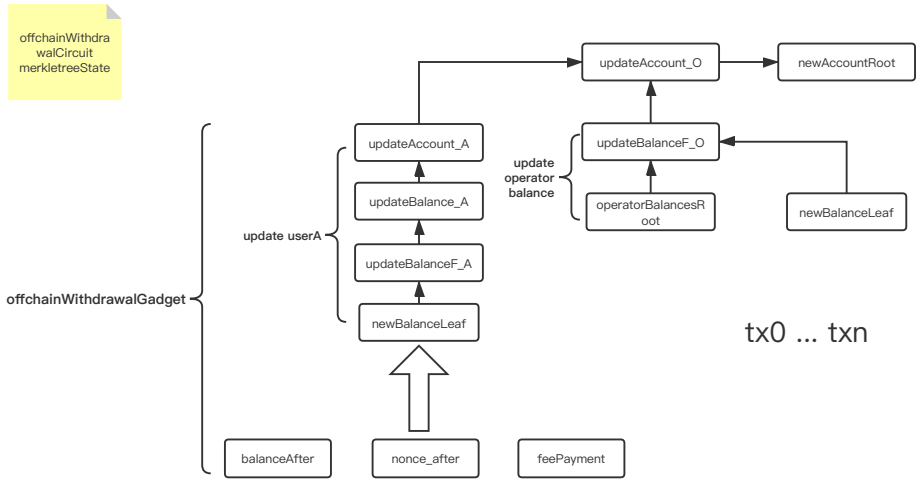


OnchainWithdrawalCircuit



OffchainWithdrawalCircuit





安比（SECBIT）实验室致力于参与共建共识、可信、有序的区块链经济体。

https://secbit.io
info@secbit.io
@secbit_io