合约方面：

新增管理的代理合约，功能包换：更改管理合约地址(用作管理合约的升级使用),

准备工作：共识节点需要对rawData={EthChainId, oldEthManagerContractAddress, newEthManagerContractAddress}进行签名，得到sigData，同时准备共识节点的publicKeyList。

接口:

updateEthCrossChainManagerContract(rawData, sigData, publicKeyList)

功能：

取旧管理合约的最新高度relayerChainBlockHeight并根据最新高度取MultiChainPublicKeys,即当前relayerchain的共识节点的公钥列表。

根据sigData及MultiChainPublicKeys对rawData进行验签，若通过，设定当前有效的EthCrossChainManager合约为newEthManagerContract

该接口需要实现的其他功能：同步txHash记录，同步验证节点变更的公钥。

从relayerChain到eth有几个问题  
Q1:当前管理合约，relayer可以跳过h=5, 提h=1, h=10的proof,交易可以正常处理，回头再提h=5的proof也可以处理，去重是通过对proof里的txHash标记实现。迁移之后，新合约需要识别旧合约里所有已被处理的txHash。

解决办法：1.可以通过把旧合约数据强制写入新管理合约解决，但可能txHash的交易有很多。2.如果旧合约不会被销毁，可通过新合约 查询旧合约的交易记录实现，看有没有被处理。这种方式会随着升级多，查询太重。

Q2:旧管理合约提交了h=1, h=10的proof,现在想通过新管理合约提交h=5的proof。如果1~10发生了共识节点变更，新合约需要有1~10的bookkeeper才能对h=5的header(含有跨链交易proof)进行验证。所以需要把旧合约的所有验证节点变更数据同步到新合约，变动不多，应该还好。

Q3: 还有一种方式，是重写管理合约，把数据和逻辑处理分开。数据为一本单独的合约，只有读和写接口，写的权限只有控制合约，relayer或业务合约调用控制合约，控制合约从数据合约取数据，传入逻辑处理合约，然后控制合约从逻辑处理合约得到返回值，根据需要写入数据合约。升级时，只对控制或逻辑合约升级，可以解决上面Q1和Q2的问题。

Eth lockProxy或业务合约变化：

当前：保存管理合约作为全局变量，每次调用直接取出调用管理合约crossChain函数。

改为：保存管代合约，每次调用管理合约之前先从管代合约中取可信管理合约地址再调用crosschain函数；跨到ETH的交易，执行unlock之前都要调用管代合约取得管理合约，比对是否是管理合约过来的请求。

Eth relayer变化：  
当前：直接根据config中的管理合约地址调用管理合约F函数  
改为：直接根据config中的管代合约先取得可信管理合约地址，再调用管理合约的F函数

ECCManagerProxy.sol: ECCMP.sol

setEthCrossChainManager( address) onlyOwner public returns (bool)

getEthCrossChainManager () return (address)

pauseEthCrossChainManager() onlyOwner public returns (bool)

unpauseEthCrossChainManager() onlyOwner public returns (bool)  
eth上的业务合约调用getEthCrossChainManager()取得EthCrossChainManager合约地址，根据IEthCrossChainManager对象调用EthCrossChainManager 合约

IECCMC.sol

Provide the interface for ECCM.sol

EthCrossChainData.sol: ECCD.sol

// put in the first place.

mapping(uint256 => bytes32) public Transactions;

mapping(uint64 => bytes) public MCblockHeaderBytes;

mapping(uint64 => bytes) public MCKeeperPubKeyBytes;

bool IsInitGenesisBlock;

mapping(bytes32 => bool) CrossChainTxExist;

uint64[] public MCKeeperHeight;

mapping(bytes32=>mapping(bytes32=>bytes) public ExtraData;

接口:

s

getHeaderBytes(uint64) public returns (bytes)

putHeaderBytes(uint64 height, bytes rawHeader) public onlyECCM returns(bool)

getMCKeeperPubKeyBytes(uint64) public return (bytes)

putMCKeeperPubKeybytes(uint64 height, bytes keepersBytes) public onlyECCM returns (bool)

getMCKeeperHeight(uint index) returns (uint64)

putMCKeeperHeight(uint64) onlyECCMC returns (bool)

getTransactions(uint256 transactionId) returns (bytes32)

putTransactions(uint256 transactionId, bytes32 transaction) onlyECCM returns (bool)

getCrossChainTxExist(bytes32 cctx) returns (bool)

putCrossChainTxExist(bytes32 cctx) onlyECCM returns(bool)

getExtraData(bytes32 key1, bytes32 key2) returns (bytes)

putExtraData(bytes32 key1, bytes32 key2, bytes value) onlyECCM returns (bool)

EthCrossChainManager: ECCM.sol

struct Header {

        uint32 version;

        uint64 chainId;

        bytes32 prevBlockHash;

        bytes32 transactionsRoot;

        bytes32 crossStatesRoot;

        bytes32 blockRoot;

        uint32 timestamp;

        uint32 height;

        uint64 consensusData;

        bytes consensusPayload;

        bytes20 nextBookkeeper;

    }

    struct ToMerkleValue {

        bytes  txHash;  // cross chain txhash

        uint64 fromChainID;

        TxParam makeTxParam;

    }

    struct TxParam {

        bytes txHash; //  source chain txhash

        bytes crossChainId;

        bytes fromContract;

        uint64 toChainId;

        bytes toContract;

        bytes method;

        bytes args;

    }

initGenesisBlock(bytes rawHeader, bytes, \_pubKeysBs) returns (bool)

changeBookKeeper(bytes rawHeader, bytes \_pubKeyBs, bytes \_sigBs) returns (bool)

syncBlockHeader(bytes rawHeader, bytes \_sigBs) public returns (bool)

verifyAndExecuteTx(bytes proof, POS[] position, bytes toMerkleValue, uint64 blockHeight) returns (bool)

syncVerifyAndExecuteTx(bytes rawHeader, bytes \_sigBs, bytes proof, POS[] position, bytes toMerkleValue, uint64 blockHeight) returns (bool)

\_executeCrossChainTx(address beCalledContract, bytes method, bytes args, bytes fromContractAddr, uint64 fromChainId) return (bool)

\_deserializeHeader(bytes) returns (Header)

\_deserializeMerkleValue(bytes) returns (ToMerkleValue)

设定全局paused状态，

pause() onlyECCMP returns (bool)

unpause() onlyECCMP returns (bool)

IEthCrossChainManager.sol: IECCM.sol

{

crossChain(uint64 \_toChainId, bytes calldata \_toContract, bytes calldata \_method, bytes calldata \_txData) external returns (bool);

}

MultiChainUtils.sol

接口：

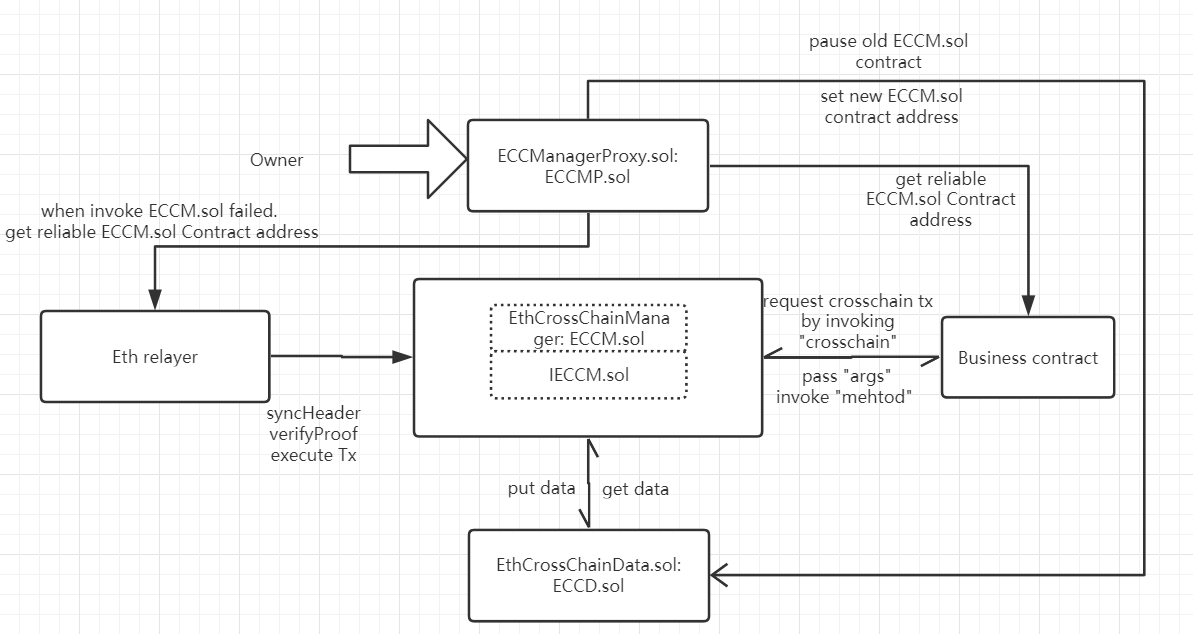
verifyMerkleProof(bytes32[] \_proof, POS[] \_position, bytes32 \_root, bytes \_toMerkleValue) returns change (bool)

verifySig (bytes \_norawHeader, bytes \_sigt, address[] \_keepers, uint \_m) returns (bool)

getBookKeeper(uint \_keyLen, uint \_m, bytes \_pubkeyListBytes) returns (bytes20, address[])

verifyPubKey(bytes \_pubKeyList) returns (bytes20, address[], uint)

\_compressMCKey(bytes key) returns (bytes newKey)



当需要升级时的操作步骤：

布署新的ECCM合约。

owner 调用ECCMP.sol 将old ECCM.sol的状态设为pause

owner调用ECCMP.sol的setEthCrossChainManager方法在ECCMP内部更新ECCM新合约地址同进调用ECCD.sol合约在ECCD合约内部更新ECCM合约地址。

注意点：

->Business contract在调用ECCM之前需要从ECCMP取得当前可信的ECCM合约hash，然后初始化IECCM对象，调用ECCM的crosschain()方法。

->当Eth relayer调用ECCM合约失败且遇到某固定错误如“contract status: paused”时，从ECCMP取得新的可信的ECCM合约Hash，保存起来，以后就用这个进行syncHeader, verifyProof, executeTx了。

->在ECCD合约内部更新ECCM合约地址这一步会确保只有新的ECCM合约才有权限将数据写入ECCD合约。”mapping(bytes32=>mapping(bytes32=>bytes) public ExtraData;”为保存字段，用来存储以后可能需要用到的数据。

->数据会存储在ECCD合约中，multichain验证proof的时候需要注意：？？？？？

<https://blog.indorse.io/ethereum-upgradeable-smart-contract-strategies-456350d0557c>

The logic contract updates the data through the setters and the data contract only allows the logic contract to call the setters. This allows the logic to be replaced while keeping the data in the same place, allowing for a fully upgradeable system.

数据合约中的数据可以随着块高变化结构可变

Manager合约可以改serialize接口逻辑，

Manager合约改变已有接口的逻辑，，，，，可以

Manager合约增加新的接口，，，，1.接口给relayer调用—可以；2.接口给业务合约调用—可以

Proxy更改manager合约地址,,,,可以

为什么不能用delegatecall？因为用的话state就会在manager或是在proxy里，这样的话数据的结构体就不能升级。

更改后：

初始化操作步骤(第一次步骤)：

Addr1布署EthCrossChainData.sol，Hash为ECCD\_Hash

Addr1 布署EthCrosssChainManager.sol，参数为ECCD\_Hash, 合约Hash为ECCM\_Hash

Addr1 布署EthCrossChainManagerProxy.sol，参数为ECCM\_Hash，合约Hash为ECCMP\_Hash

Addr1 调用ECCD\_Hash的transferOwnership方法，input为ECCM\_Hash

Addr1调用ECCM\_Hash的transferOwnership方法，input为ECCMP\_Hash

环境设定ok，可以正常使用

迁移操作步骤：

Addr1调用ECCMP\_HASH的pauseEthCrossChainManager()函数，让ECCMP\_Hash的合约停止运行,让ECCM\_Hash的合约停止运行，让ECCD\_Hash的合约停止put。

Addr1布署新的EthCrosssChainManager.sol合约，可以增加新函数，可以更改逻辑，可以新增函数，可以更新已有结构体，Hash为ECCM\_Hash\_new.

Addr1调用ECCM\_Hash\_new合约的transferOwnership方法，input为ECCMP\_Hash

Addr1调用ECCMP\_Hash的upgradeEthCrossChainManager()方法，input为ECCM\_Hash\_new。

迁移操作完成，可以正常使用。

Addr1 调用ECCMP\_Hash的unpauseEthCrossChainManager()函数,