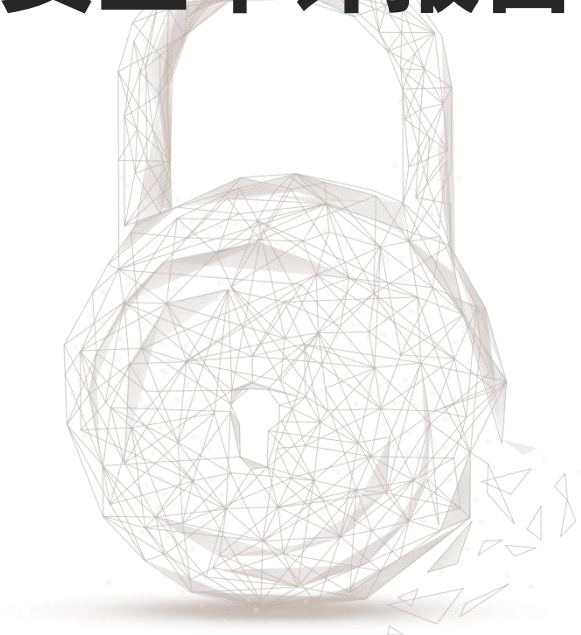
成都链安
BEOSIN

EOS 智能合约

安全审计报告



审计编号：201905161843

审计合约名称：

ipsecontract

审计合约地址：

ipsecontract

审计合约源代码 hash（sha256）：

777140ea1d208efaf11561e12bfe22a558a2a83042d5f29a7323f573e52f929a

合约审计开始日期：2019.04.28

合约审计完成日期：2019.05.16

审计结果：通过（优）

审计团队：成都链安科技有限公司

审计类型及结果：

序号	审计类型	审计子项	审计结果
1	代码规范审计	基本格式规范审计	通过
		命名约束规范审计	通过
		变量初始化审计	通过
2	函数调用审计	函数调用权限审计	通过
		apply 安全审计	通过
		系统 API 函数使用审计	通过
		inline action 安全审计	通过
		deferred action 安全审计	通过
		事件响应安全审计	通过
3	整型溢出审计	-	通过
4	内存溢出审计	数组越界审计	通过
		指针异常使用审计	通过
		栈溢出审计	通过
		堆溢出审计	通过
5	随机数安全审计	-	通过

6	数据存储审计	RAM 消耗审计	通过
		存储安全审计	通过
7	业务逻辑审计	交易顺序安全审计	通过
		业务限制绕过审计	通过
		逻辑实现审计	通过

备注：审计意见及建议请见代码注释。

审计结果说明：

本公司采用形式化验证、静态分析、动态分析、典型案例测试和人工审核的方式对智能合约ipsecontract的代码规范性、安全性以及业务逻辑三个方面进行多维度全面的安全审计。**经审计，ipsecontract合约通过所有检测项，合约审计结果为通过(优)，合约可正常使用。**

其他审计建议：

1、 调用issue函数发行POST代币时，对代币issuer的发行量可能会超过系统设置的上限。

问题描述：如图1所示，合约逻辑要求对代币issuer的发行量不能超过代币最大发行量的30%，但是由于其判定的依据是当前发行量，所以，在某些特殊的情况下，可使得对issuer的发行量超过30%。例如：第一次发行代币时，可全部发行给issuer，因为代币当前发行量为0。

```
222     require_auth(st.issuer);
223     if(to==st.issuer){
224         eosio_assert(st.supply.amount < st.max_supply.amount*0.3,"you could not issuer token number over 0.3 ratio");
225     }
226     eosio_assert(quantity.is_valid(), "invalid quantity");
```

图1 issuer函数部分源码

审计建议：以本次的发行结果为判断依据。

修复结果：线下进行限制。经项目方确认，在实际业务中，项目方会首先对issuer发行略少于30%的代币。

2、 未使用的合约代码

问题描述：如图2、图3所示，合约中存在部分函数、结构体、变量未被使用。

```
111  struct transfer_args {
112      account_name  from;
113      account_name  to;
114      asset         quantity;
115      std::string        memo;
116  };
117
118  void apply(account_name contract,action_name act);
```

图2 合约中未被使用的结构体和函数

```
680  /// @abi action
681  void findreviewby(uint64_t count){
682      account_name name = eosio::string_to_name("ipsecontract");
683      auto countindex = _reviews.get_index<N(bycount)>();
684      auto cou = countindex.find(count);
685      for(;cou!=countindex.end();cou++){
686          print("address:",cou->address," count:",cou->count);
687      }
688  };
```

图3 合约中未被使用的变量

审计建议：删除该部分未被使用的代码。

修复结果：忽略。经项目方确认，这部分代码不影响合约功能，可忽略。

3、addreview函数无调用限制，存在恶意刷票的可能

问题描述：如图4所示的addreview函数是用来评议目标账户，但无评议次数和目标的限制，因此可能存在用户恶意刷票的情况。

```
624  /// @abi action
625  void addreview(account_name address,account_name selfaddress){
626      eosio_assert(is_account(address),"address account does not exist");
627      account_name name = eosio::string_to_name("ipsecontract");
628      auto blacklist_iterator = _blacklists.find(address);
629      if(blacklist_iterator!=_blacklists.end()){
630          print("the address you review was in the blacklist");
631      }else{
632          auto blacklist_iterator_ = _blacklists.find(selfaddress);
633          if(blacklist_iterator_!=_blacklists.end()){
634              print("you are in the blacklist!");
635          }else{
636              require_auth(selfaddress);
637              auto itr = _reviews.find(address);
638              if(itr!=_reviews.end()){
639                  _reviews.modify(itr,name,[&](auto& p){
640                      p.count++;
641                  });
```

图4 addreview函数部分源代码截图

审计建议：添加评议目标和次数的限制或者线下监控。

修复结果：忽略。经项目方确认，项目方会在线下对评议行为进行监控，并及时将恶意用户拉入黑名单。

4、存在部分黑名单账户可取出全部抵押的情形

问题描述：如图5、图6所示，调用modifyblacklist函数和addblack函数去修改或设置账户的黑名单状态移除时间时，并未对黑名单状态移除时间大于100年(距离当前时间戳)的账户进行处理。这可能导致出现这样的问题：当设置已参与抵押账户的黑名单状态移除时间大于100年后，再去调用如图7所示的eraseblackli函数将该账户从黑名单中移除时，会修改其挖矿总奖励(totalasset)为0。如果这时该黑名单账户再去调用reducepledge函数提取抵押，会因为如图8所示的计算逻辑而能够提取出全部抵押。

```
385     /// @abi action
386     void modifyblackl(account_name address,uint64_t blocktime){
387         account_name name = eosio::string_to_name("ipsecontract");
388         require_auth(name);
389         auto& blacklist = _blacklists.get(address,"the address you modify is not in the blacklist!");
390         _blacklists.modify(blacklist,get_self(),[&](auto& p){
391             p.blocktime = blocktime;
392         });
393         print("address:",address," modified successfully");
394     }
```

图5 modifyblacklist函数源代码

```
701     void addblack(account_name address,uint64_t blocktime){
702         account_name name = eosio::string_to_name("ipsecontract");
703         require_auth(name);
704         if(blocktime>(current_time()/1000)){
705             auto blacklist_iterator = _blacklists.find(address);
706             if(blacklist_iterator!=_blacklists.end()){
707                 _blacklists.modify(blacklist_iterator,get_self(),[&](auto& p){
708                     p.blocktime = blocktime;
709                 });
710                 print("update the address in blacklist scucess!");
711             }else{
712                 _blacklists.emplace(name,[&](auto &p){
713                     p.address = address;
714                     p.blocktime = blocktime;
715                 });
716                 print("add blacklist element success!");
717             }
718         }else{
719             print("blocktime must over now!");
720         }
721     }
```

图6 addblack函数源代码

```
358     /// @abi action
359     void eraseblackli(account_name address){
360         account_name name = eosio::string_to_name("ipsecontract");
361         require_auth(name);
362         auto blacklist_iterator = _blacklists.find(address);
363         if(blacklist_iterator!=_blacklists.end()){
364             auto& black = _blacklists.get(address);
365             auto blocktime = black.blocktime;
366             if(blocktime > (current_time()/1000 + 3153600000000)){
367                 auto pledgeitr = _pledges.find(address);
368                 if(pledgeitr!=_pledges.end()){
369                     auto& pledge =  pledges.get(address);
370                     extended_asset _totalasset_(0,S(4,POST));
371                     _totalasset_.contract = name;
372                     _pledges.modify(pledgeitr,name,[&](auto& p){
373                         p.totalasset = _totalasset_;
374                     });
375                     print("you could minging again!");
```

图7 eraseblackli函数源代码

```
422     void reducepledge(account_name to,asset quantity){
423         require_auth(to);
424         eosio_assert(quantity.is_valid(),"invalid quantity");
425         eosio_assert(quantity.amount>0,"must withdraw positive quantity");
426         eosio_assert(quantity.symbol==S(4, POST),"only POST token allowed");
427         account_name name = eosio::string_to_name("ipsecontract");
428         // eosio_assert(quantity.contract==name,"the toekn issue contract must be
429         auto pledgeitr = _pledges.find(to);
430         if(pledgeitr!=_pledges.end()){
431             //auto addtime = current_time();
432             auto& pledges = _pledges.get(to);
433             auto inittime = pledges.addtime;
434             float pledgeratio = compute_pledgeratio(inittime);
435             extended_asset pledgeasset = pledges.pledgeasset;
436             extended_asset totalasset = pledges.totalasset;
437             // user can withdraw pledge:  pledgeasset - totalasset*pledgeratio
438             double totalpledge = totalasset.amount*pledgeratio;
439             // totalpledge *= std::pow(10,4);
440             auto could_withdraw = pledgeasset.amount - totalpledge;
441             auto balance = pledgeasset.amount;
442             if(quantity.amount>could_withdraw){
443                 print("withdraw over the could withdraw:",could_withdraw);
```

图8 reducepledge函数计算可提取抵押数量部分代码

审计建议：在调用eraseblackli函数将账户从黑名单移除时，将对应的挖矿总奖励(totalasset)置为0时，将其对应的当前抵押资产(pledgeasset)也同步置为0。

修复结果：忽略。

合约源代码审计注释：

```
#include <eosiolib/eosio.hpp>
#include <eosiolib/asset.hpp>
#include <eosiolib/print.hpp>
#include <eosiolib/dispatcher.hpp>
#include <eosiolib/multi_index.hpp>
#include <cmath>
using namespace eosio;
namespace eosiosystem {
   class system_contract;
}

class ipsecontract : public contract{
private:
    // file category  0:html  1:image  2:video  3:audio   4:dir   5:package
6:other
    // 成都链安 // 文件类型与其单位算力映射
    const std::map<uint8_t,uint8_t> file = {
        {0,20},
        {1,4},
        {2,2},
        {3,15},
        {4,1},
        {5,5},
        {6,8}
    };

    // create the multi index tables to store the data
    /// @abi table
    struct account
    {
        // uint64_t id;
        asset    balance; // 成都链安 // 代币余额
        uint64_t primary_key() const {return balance.symbol.name();}
    };
    typedef eosio::multi_index<N(accounts),account> accounts;

    /// @abi table
    struct stat
    {
        // uint64_t       id;
        asset          supply; // 成都链安 // 代币当前发行量
        asset          max_supply; // 成都链安 // 代币最大发行量
        account_name   issuer; // 成都链安 // 代币发行者
        uint64_t primary_key() const {return  supply.symbol.name();}
    };
```

```
    typedef eosio::multi_index<N(stat),stat> stats;

    /// @abi table
    // 成都链安 // 系统挖矿信息表
    struct globalvar
    {
        uint64_t    key; // 成都链安 // 主键
        uint64_t    totalworkforce; // 成都链安 // 总算力
        uint64_t    addworkforce; // 成都链安 // 增加算力
        uint64_t    settletime; // 成都链安 // 结算时间
        uint64_t    increasetime; // 成都链安 // 增加时间
        extended_asset  balance; // 成都链安 // 发行资产
        uint64_t primary_key() const {return key;}
    };
    typedef eosio::multi_index<N(globalvar),globalvar> globalvars;

    /// @abi table review
    struct review
    {
        account_name    address;//primary key
        uint64_t count; // 成都链安 // 被评议次数
        auto primary_key() const {return address;}
        uint64_t get_count() const {return count;}
    };
    typedef
eosio::multi_index<N(review),review,indexed_by<N(bycount),const_mem_fun<review,uint6
4_t,&review::get_count>>> reviews;

    /// @abi table reviewer
    struct reviewer
    {
        account_name    address;//primary key
        uint64_t count; // 成都链安 // 评议次数
        auto primary_key() const {return address;}
        uint64_t get_count() const {return count;}
    };
    typedef
eosio::multi_index<N(reviewer),reviewer,indexed_by<N(bycount),const_mem_fun<reviewer
,uint64_t,&reviewer::get_count>>> reviewers;

    /// @abi table blacklist
    struct blacklist
    {
        account_name address;
        uint64_t blocktime; // 成都链安 // 黑名单状态移除时间戳
```

```
        auto primary_key() const {return address;}
    };
    typedef eosio::multi_index<N(blacklist),blacklist> blacklists;

    /// @abi table pledge
    struct pledge
    {
        account_name    address;
        extended_asset    pledgeasset; // 成都链安 // 当前抵押资产
        extended_asset    totalasset; // 成都链安 // 挖矿总奖励
        uint64_t    addtime;
        auto primary_key() const {return address;}
    };
    typedef eosio::multi_index<N(pledge),pledge> pledges;

    // local instances of the multi indexes.
    globalvars _globalvars;
    reviews _reviews;
    reviewers _reviewers;
    blacklists _blacklists;
    pledges _pledges;
    accounts accountstable;
    stats statstable;

public:
    // using contract::contract;
    ipsecontract(account_name
self):contract(self),_globalvars(self,self),_reviews(self,self),_reviewers(self,self
),_blacklists(self,self),_pledges(self,self),accountstable(self,self),statstable(sel
f,self) {}
    struct transfer_args { // 成都链安 // 未被使用的结构体，可删除
        account_name  from;
        account_name  to;
        asset         quantity;
        std::string       memo;
    };

    void apply(account_name contract,action_name act); // 成都链安 // 未被实现的函数接
口，可删除
    // 成都链安 // 获取指定代币的供应量，内部函数，未被使用
    asset get_supply(symbol_name sym) const {
        stats statstable(_self, sym);
        const auto& st = statstable.get(sym);
        return st.supply;
    }
    // 成都链安 // 获取指定账户的代币余额，内部函数，未被使用
```

```
    asset get_balance(account_name owner,symbol_name sym) const {
        accounts accountstable(_self, owner);
        const auto& ac = accountstable.get(sym);
        return ac.balance;
    }

    /// @abi action
    // 成都链安 // 创建代币，设置代币基本信息和代币 issuer
    void create(account_name issuer,asset maximum_supply){
        require_auth(_self); // 成都链安 // 调用权限检查
        auto sym = maximum_supply.symbol;
        // 成都链安 // 检查新增代币参数的有效性
        eosio_assert(sym.is_valid(), "invalid symbol name");
        eosio_assert(maximum_supply.is_valid(), "invalid supply");
        eosio_assert(maximum_supply.amount > 0, "max-supply must be positive");

        stats statstable(_self, sym.name());
        auto existing = statstable.find(sym.name());
        eosio_assert(existing == statstable.end(),"token with symbol already
exists" ); // 成都链安 // 要求新增代币不存在于代币资产表中
        print("name:",sym.name(),"\n");
        print("symbol:",maximum_supply.symbol,"\n");
        print("maximum_supply:",maximum_supply,"\n");
        print("issuer:",issuer,"\n");
        // 成都链安 // 创建代币，设置代币基本信息
        statstable.emplace(_self, [&](auto& s){
            // s.id = sym.name();
            s.supply.symbol = maximum_supply.symbol;
            s.max_supply    = maximum_supply;
            s.issuer        = issuer;
        });
    }
    // 成都链安 // 获取当前时间戳的可增发代币数量
    uint64_t supplytoken(){
        auto inittime = 1546271999000;  //1443143890000
        auto now = current_time()/1000;
        auto year = int((now-inittime)/1000/365/24/60/60); // 成都链安 // 计算当前时间戳
与初始时间的年份差
        if(year<=2){
            return 47945200000;
        }else if (year<=4){
            return 23972600000;
        }else if(year<=6){
            return 11986300000;
        }else if(year<=8){
            return 5993100000;
```

```
        }else{
            return 299650000;
        }
    };

    /// @abi action
    // 成都链安 // 增加代币最大发行量
    void increase(asset increase_supply){
        require_auth(_self); // 成都链安 // 调用权限检查
        auto globalvarsitr = _globalvars.find(0);
        if(globalvarsitr!=_globalvars.end()){
            auto& globalvars = _globalvars.get(0);
            uint64_t now = current_time();
            uint64_t increasetime = globalvars.increasetime;
            if((now-31536000000000)>increasetime){ // 成都链安 // 要求与上次的增发时间之差必须大于 1 年
                auto increase_total = increase_supply.amount; // 成都链安 // 获取增发数量
                auto increase_supply_ = supplytoken(); // 成都链安 // 调用 supplytoken 函数获取当前最大可增发数量
                if(increase_total>increase_supply_){ // 成都链安 // 检查本次代币增发的有效性，要求增发数量不能超过当前最大可增发数量
                    print("you could not increase supply over the limit:",increase_supply_);
                }else{
                    auto sym = increase_supply.symbol;
                    stats statstable(_self, sym.name());
                    auto statstableitr = statstable.find(sym.name());
                    if(statstableitr == statstable.end()){ // 成都链安 // 检查增发代币是否存在
                        print("token with symbol not created!");
                    }else{
                        auto& statstable_ = statstable.get(sym.name());
                        asset _max_supply = statstable_.max_supply; // 成都链安 // 获取对应代币的当前最大发行量
                        asset max_supply(_max_supply.amount+increase_supply.amount,S(4,POST)); // 成都链安 // 计算增发后的代币最大发行量
                        statstable.modify(statstableitr,_self,[&](auto& s){
                            s.max_supply = max_supply; // 成都链安 // 更新对应代币最大发行量
                        });
                        _globalvars.modify(globalvarsitr,_self,[&](auto& g){
```

```
                        g.increasetime = now; // 成都链安 // 更新代币增发时间
                    });
                }
            }
        }else{
            print("increase supply only once a year,if you want increase supply
then you could in next year!");
        }
    }else{
        print("the global var is not exits!");
    }
}

/// @abi action
// 成都链安 // 发行代币
void issue(account_name to,asset quantity,std::string memo){
    auto sym = quantity.symbol;
    eosio_assert(sym.is_valid(), "invalid symbol name");
    eosio_assert(memo.size() <= 256, "memo has more than 256 bytes");

    auto sym_name = sym.name();
    stats statstable(_self, sym_name);
    auto existing = statstable.find(sym_name);
    eosio_assert(existing != statstable.end(), "token with symbol does not
exist, create token before issue");
    const auto& st = *existing;
    require_auth(st.issuer); // 成都链安 // 调用权限检查
    // 成都链安 // 限制对 issuer 的发行量必须少于最大发行量的 30%
    // 成都链安 // 实际操作时，可向 issuer 发行超过最大发行量的 30%，例如：第一次就发行全
部的代币给 issuer 自己
    // 成都链安 // 建议以本次发行的结果作为数值上的判断依据，或者线下进行限制
    if(to==st.issuer){
        eosio_assert(st.supply.amount < st.max_supply.amount*0.3,"you could not
issuer token number over 0.3 ratio");// issuer to self < 30%
    }
    // 成都链安 // 检查发行代币参数的有效性
    eosio_assert(quantity.is_valid(), "invalid quantity");
    eosio_assert(quantity.amount > 0, "must issue positive quantity");

    eosio_assert(quantity.symbol == st.supply.symbol, "symbol precision
mismatch");
    eosio_assert(quantity.amount <= st.max_supply.amount - st.supply.amount,
"quantity exceeds available supply"); // 成都链安 // 要求本次发行量不能超过当前可发行量
    // 成都链安 // 修改对应代币的当前供应量
    statstable.modify(st,_self,[&](auto& s) {
```

```
            s.supply += quantity;
        });

        add_balance(st.issuer, quantity, st.issuer); // 成都链安 // 调用内部函数 add_balance 修改代币 issuer 的代币余额

        if(to != st.issuer) {
            SEND_INLINE_ACTION( *this, transfer, {st.issuer,N(active)}, {st.issuer, to, quantity, memo}); // 成都链安 // issuer 向目标地址 to 发送代币
        }
    }

    /// @abi action
    // 成都链安 // 代币转账函数
    void transfer(account_name from,account_name to,asset quantity,std::string memo){
        eosio_assert(from != to,"cannot transfer to self"); // 成都链安 // 不能向自己进行代币转账
        require_auth(from); // 成都链安 // 调用权限检查
        eosio_assert(is_account(to),"to account does not exist"); // 成都链安 // 要求目标地址 to 必须是有效地址
        auto sym = quantity.symbol.name();
        stats statstable(_self, sym);
        const auto& st = statstable.get(sym);
        // 成都链安 // 向转账双方发送转账通知
        require_recipient(from);
        require_recipient(to);
        // 成都链安 // 检查代币参数的有效性
        eosio_assert(quantity.is_valid(), "invalid quantity" );
        eosio_assert(quantity.amount > 0, "must transfer positive quantity" );
        eosio_assert(quantity.symbol == st.supply.symbol, "symbol precision mismatch" );
        eosio_assert(memo.size() <= 256, "memo has more than 256 bytes" );

        sub_balance(from,quantity); // 成都链安 // 调用内部函数 sub_balance 减少 from 的代币余额
        add_balance(to,quantity,from); // 成都链安 // 调用内部函数 add_balance 增加转账目标地址 to 的代币余额

        // if someone transfer POST to ipsecontract,then add pledge
        // 成都链安 // 检查转账双方，如果转账发起地址是本合约，或者目标地址不是本合约，则结束转账操作
        if(from==_self||to!=_self){
            return;
```

```
        }

        if(quantity.symbol==S(4,POST)){ // 成都链安 // 检查代币类型，如果转账的代币是POST，则进行相应的抵押操作
            auto pledgeitr = _pledges.find(from);
            if(pledgeitr!=_pledges.end()){ // 成都链安 // 要求转账发起者必须在抵押表中
                auto& pledges = _pledges.get(from); // 成都链安 // 获取对应的抵押记录
                extended_asset pledgeasset = pledges.pledgeasset;
                extended_asset totalasset = pledges.totalasset;
                auto balance = pledgeasset.amount; // 成都链安 // 获取该用户的当前抵押资产数量
                extended_asset _pledgeasset_(balance+quantity.amount,S(4,POST)); // 成都链安 // 计算该地址本次抵押后的抵押资产
                _pledgeasset_.contract = _self;
                _pledges.modify(pledgeitr,_self,[&](auto& p){ // 成都链安 // 更新抵押表对应的抵押记录信息
                    p.pledgeasset = _pledgeasset_;
                });
            }else{ // 成都链安 // 如果转账发起者不在抵押表中，会出现提示信息，但不会对转账代币进行退款
                print("can't find the address in pledge table.");
            }
        }
    }
    // 成都链安 // 内部函数，减少指定地址的代币余额
    void sub_balance(account_name owner,asset value){
        accounts from_acnts(_self, owner);
        const auto& from = from_acnts.get( value.symbol.name(), "no balance object found");
        eosio_assert(from.balance.amount >= value.amount, "overdrawn balance"); // 成都链安 // 减少数额检查，要求不能超过指定地址的当前余额

        if( from.balance.amount == value.amount) { // 成都链安 // 如果减少后该地址的代币余额为 0，则从对应表中移除该地址的记录
            from_acnts.erase(from);
        } else { // 成都链安 // 否则，更新该地址的代币余额
            from_acnts.modify(from, owner, [&]( auto& a){
                a.balance -= value;
            });
        }
    }
    // 成都链安 // 增加指定地址的代币余额
    void add_balance(account_name owner, asset value, account_name ram_payer){
```

```
        accounts to_acnts(_self,owner);
        auto to = to_acnts.find(value.symbol.name());
        if(to == to_acnts.end()){ // 成都链安 // 如果指定地址不在资产表中，则新增记录，存储该地址代币余额
            to_acnts.emplace( ram_payer, [&]( auto& a){
                // a.id = value.symbol.name();
                a.balance = value;
            });
        }else{ // 成都链安 // 否则，更新该地址的代币余额
            to_acnts.modify(to,_self, [&]( auto& a){
                a.balance += value;
            });
        }
    }

    /// @abi action
    // 成都链安 // 查询指定地址的抵押信息
    void findpledge(account_name address){
        auto& pledge = _pledges.get(address,"address does't exists in the pledge table");
        print(pledge.address," ",pledge.pledgeasset," ",pledge.totalasset," ",pledge.addtime);
    }

    /// @abi action
    // 成都链安 // 添加指定地址至黑名单，并设置其黑名单状态的移除时间
    void addblacklist(account_name address,uint64_t blocktime){
        account_name name = eosio::string_to_name("ipsecontract");
        require_auth(name); // 成都链安 // 调用权限检查
        auto blacklist_iterator = _blacklists.find(address);
        if(blacklist_iterator == _blacklists.end()){
            _blacklists.emplace(get_self(),[&](auto &p){ // 成都链安 // 如果指定的地址不在黑名单中，则新增记录，并初始化该地址的黑名单数据
                p.address = address;
                p.blocktime =  blocktime;
            });
        }
        if(blocktime > (current_time()/1000 + 3153600000000)){ // 成都链安 // 如果设置的黑名单状态移除时间距离当前时间大于 100 年，则没收其抵押的 POST
            auto pledgeitr = _pledges.find(address);
            if(pledgeitr!=_pledges.end()){ // 成都链安 // 如果对应地址存在于抵押表中，则更新抵押数据
                auto& pledge = _pledges.get(address);
                extended_asset pledgeasset = pledge.pledgeasset;
                extended_asset _pledgeasset_(0,S(4,POST));
```

```
                _pledgeasset_.contract = name;
                _pledges.modify(pledgeitr,name,[&](auto& p){
                    p.pledgeasset = _pledgeasset_; // 成都链安 // 更新抵押资产为 0
                });
                print("confiscate POST from pledge table
success:",pledgeasset.amount,"\n");
            }
        }
        print("add address to blacklist successfully ");
    };

    /// @abi action
    // 成都链安 // 查询指定地址的黑名单状态信息
    void findblacklis(account_name address){
        auto blacklistitr = _blacklists.find(address);
        if(blacklistitr!=_blacklists.end()){
            auto& user = _blacklists.get(address);
             print(user.address," ",user.blocktime);
        }else{
            print("address doesn't exist in the blacklist");
        }
    };

    /// @abi action
    // 成都链安 // 将指定地址从黑名单移除
    void eraseblackli(account_name address){
        account_name name = eosio::string_to_name("ipsecontract");
        require_auth(name); // 成都链安 // 调用权限检查
        auto blacklist_iterator = _blacklists.find(address);
        if(blacklist_iterator!=_blacklists.end()){ // 成都链安 // 当指定地址存在于黑名单
中
            auto& black = _blacklists.get(address);
            auto blocktime = black.blocktime; // 成都链安 // 获取对应地址的黑名单状态移除
时间
            if(blocktime > (current_time()/1000 + 3153600000000)){ // 成都链安 // 如果其
移除时间距离当前时间大于 100 年，则将该地址的挖矿总奖励置为 0
                auto pledgeitr = _pledges.find(address);
                if(pledgeitr!=_pledges.end()){ // 成都链安 // 如果对应地址存在于抵押表中，
则更新抵押数据
                    auto& pledge = _pledges.get(address);
                    extended_asset _totalasset_(0,S(4,POST));
                    _totalasset_.contract = name;
                    _pledges.modify(pledgeitr,name,[&](auto& p){
                        p.totalasset = _totalasset_; // 成都链安 // 更新挖矿总奖励为 0
                    });
```

```
                    print("you could minging again!");
                }
            }
            _blacklists.erase(blacklist_iterator); // 成都链安 // 移除该地址在黑名单中的记录
            print("address:",address," deleted successfully");
        }else{
            print("the address you delete is not in the blacklist!");
        }
    }

    /// @abi action
    // 成都链安 // 修改指定地址的黑名单状态移除时间
    // 成都链安 // 注意：该函数不会对当前时间戳与黑名单状态移除时间之差大于 100 年的地址进行处理
    void modifyblackl(account_name address,uint64_t blocktime){
        account_name name = eosio::string_to_name("ipsecontract");
        require_auth(name); // 成都链安 // 调用权限检查
        auto& blacklist = _blacklists.get(address,"the address you modify is not in the blacklist!"); // 成都链安 // 获取对应地址的黑名单记录
        _blacklists.modify(blacklist,get_self(),[&](auto& p){
            p.blocktime = blocktime; // 成都链安 // 修改对应的移除时间
        });
        print("address:",address," modified successfully");
    }

    /// @abi action
    // 成都链安 // 获取当前合约版本
    void version(std::string version){
        print("poseidon chain version:0.1.1");
    };
    // 成都链安 // 计算抵押比例
    auto compute_pledgeratio(uint64_t inittime){
        auto now = current_time(); // 成都链安 // 获取当前时间戳
        auto year = int((now-inittime)/1000/365/24/60/60/1000); // 成都链安 // 获取当前时间戳与初始时间的年份差
        if(year>5){
            year = 5;
        }
        return 0.5-0.1*year;
    }
    // 成都链安 // 计算抵押是否足够
    auto judge_pledge(extended_asset pledgeasset,extended_asset totalasset,float pledgeratio){
```

```
        auto pledgeamount = pledgeasset.amount; // 成都链安 // 抵押数量
        auto totalamount = totalasset.amount; // 成都链安 // 挖矿总奖励
        float ratio = (float) pledgeamount/totalamount; // 成都链安 // 计算当前的抵押比例
        if(totalamount==0 || ratio>=pledgeratio){ // 成都链安 // 要求总抵押数量为 0，或者当前抵押比例大于输入的比例
            return true;
        }else{
            return false;
        }
    }

    /// @abi action
    // 成都链安 // 提取抵押
    void reducepledge(account_name to,asset quantity){
        require_auth(to); // 成都链安 // 调用权限检查
        // 成都链安 // 检查提取抵押的参数有效性
        eosio_assert(quantity.is_valid(),"invalid quantity");
        eosio_assert(quantity.amount>0,"must withdraw positive quantity");
        eosio_assert(quantity.symbol==S(4, POST),"only POST token allowed");
        account_name name = eosio::string_to_name("ipsecontract");
        // eosio_assert(quantity.contract==name,"the toekn issue contract must be ipsecontract");
        auto pledgeitr = _pledges.find(to);
        if(pledgeitr!=_pledges.end()){ // 成都链安 // 当指定地址存在于抵押表中
            //auto addtime = current_time();
            auto& pledges = _pledges.get(to);
            auto inittime = pledges.addtime;
            float pledgeratio = compute_pledgeratio(inittime); // 成都链安 // 调用 compute_pledgeratio 函数获取抵押比例
            // 成都链安 // 获取地址 to 的抵押资产和挖矿总奖励
            extended_asset pledgeasset = pledges.pledgeasset;
            extended_asset totalasset = pledges.totalasset;
            // user can withdraw pledge:  pledgeasset - totalasset*pledgeratio
            double totalpledge = totalasset.amount*pledgeratio; // 成都链安 // 计算挖矿总奖励数量
            // totalpledge *= std::pow(10,4);
            auto could_withdraw = pledgeasset.amount - totalpledge; // 成都链安 // 计算用户可提取的抵押资产数量
            auto balance = pledgeasset.amount;
            if(quantity.amount>could_withdraw){
                print("withdraw over the could withdraw:",could_withdraw);
            }else{ // 成都链安 // 要求用户提取的抵押资产数量不超过可提取的数量
                extended_asset _pledgeasset_(balance-quantity.amount,S(4,POST)); //
```

```
成都链安 // 计算新的抵押资产
                _pledgeasset_.contract = name;
                _pledges.modify(pledgeitr,name,[&](auto& p){
                    p.pledgeasset = _pledgeasset_; // 成都链安 // 更新抵押资产
                });
                action(
                    permission_level{N(ipsecontract),N(active)},
                    N(ipsecontract),N(issue),
                    std::make_tuple(to,quantity,std::string("withdraw POST from the
pledge table"))
                ).send(); // 成都链安 // 调用 issue 函数向目标地址 to 发行代币
                print("withdraw POST from pledge table success:",quantity.amount);
            }
        }else{ // 成都链安 // 如果指定地址不存在于抵押表中，新增该地址的抵押记录
            extended_asset amount(0,S(4,POST));
            amount.contract = name;
            auto addtime = current_time();
            _pledges.emplace(name,[&](auto &p){ // 成都链安 // 新增抵押记录
                p.address = to;
                p.pledgeasset = amount;
                p.totalasset = amount;
                p.addtime = addtime;
            });
            print("you are not in pledge table,insert into table success!");
        }
    }

    /// @abi action
    // 成都链安 // 添加挖矿，系统的第一次挖矿不会产生抵押记录信息，只初始化挖矿参数
    void addlabel(std::string label_,std::string hash,account_name address,uint8_t
category,uint64_t size){
        account_name name = eosio::string_to_name("ipsecontract");
        require_auth(name); // 成都链安 // 调用权限检查
        auto bitr = _blacklists.find(address); // judge the address from blacklist
        if(bitr!=_blacklists.end()){ // 成都链安 // 如果指定地址处于黑名单中，则进行黑名单
状态移除时间检查
            auto& blacklist = _blacklists.get(address);
            auto blocktime = blacklist.blocktime;
            if(blocktime>(current_time()/1000)){ // 成都链安 // 如果该地址未到达黑名单状
态移除时间
                 print("the address:",address," in the blacklist");
            }else{
                _blacklists.erase(bitr); // 成都链安 // 如果已到达移除时间，则将该地址从黑
名单中移除，但并不会记录挖矿结果
            }
```

```
        }else{ // 成都链安 // 对应的账户不存于黑名单中
            // judge the address from the pledge
            extended_asset zero(0,S(4,POST));
            zero.contract = name;
            extended_asset pledgeasset = zero;
            extended_asset totalasset = zero;
            float pledgeratio = 0.5;
            bool pledgeflag = true;
            auto pledgeitr = _pledges.find(address);
            if(pledgeitr!=_pledges.end()){ // 成都链安 // 当指定地址存在于抵押表中
                auto& pledges = _pledges.get(address);
                // 成都链安 // 更新抵押数据
                pledgeasset = pledges.pledgeasset;
                totalasset = pledges.totalasset;
                auto addtime = pledges.addtime;
                pledgeratio = compute_pledgeratio(addtime); // 成都链安 // 调用
compute_pledgeratio 函数计算抵押比例
                pledgeflag = judge_pledge(pledgeasset,totalasset,pledgeratio); // 成
都链安 // 调用 judge_pledge 函数判断抵押是否充足
            }
            if(pledgeflag){
                if(category>=0 && size>0 && category<7 && !hash.empty()){ // 成都链安
// 如果抵押充足，则检查挖矿的输入参数
                    // compute the workforce for the label
                    auto force = compute_workforce(category,size); // 成都链安 // 调用
compute_workforce 函数计算算力
                    auto globalvar_itr =  _globalvars.find(0);
                    if(globalvar_itr != _globalvars.end()){ // 成都链安 // 如果挖矿信息
表存在
                        auto& globalvars = _globalvars.get(0, "there is no vars in
the globalvars table");
                        // 成都链安 // 获取对应表的基本信息
                        uint64_t settletime = globalvars.settletime;
                        uint64_t totalworkforce = globalvars.totalworkforce;
                        uint64_t addworkforce = globalvars.addworkforce;
                        extended_asset balance = globalvars.balance;
                        double decayrate = 1.0; // 成都链安 // 初始衰减率为 1
                        // 成都链安 // 根据当前算力情况更新衰减率
                        if(addworkforce>10*totalworkforce){
                            decayrate = 0.01;
                        }
                        else if (addworkforce>8*totalworkforce)
                        {
                            decayrate = 0.03;
```

```
                        }
                        else if (addworkforce>6*totalworkforce)
                        {
                            decayrate = 0.06;
                        }
                        else if (addworkforce>4*totalworkforce)
                        {
                            decayrate = 0.09;
                        }
                        else if (addworkforce>2*totalworkforce)
                        {
                            decayrate = 0.12;
                        }

                        // ----------------------------------------settle---------
---------------------------------------
                        auto amount_ = balance.amount;
                        auto symbol = balance.symbol;
                        double sharetoken_ =  (double)
force/totalworkforce*amount_*decayrate; // 成都链安 // 计算本次挖矿总奖励数量
                        uint64_t sharetoken = std::floor(sharetoken_*(1-
pledgeratio)); // 成都链安 // 计算分发给用户的代币数量
                        // sharetoken *= std::pow(10,4);
                        extended_asset amount((sharetoken),S(4,POST)); // 成都链安 //
获取分发给挖矿者的 POST 资产
                        // amount.contract = name;
                        print(force," ",totalworkforce," ",amount_," ",sharetoken_,"
",sharetoken," ",amount,"\n");
                        // sharetoken_ *= std::pow(10,4);
                        // 成都链安 // 获取本次挖矿总资产
                        extended_asset
allminingasset(std::round(sharetoken_),S(4,POST));  // allminingasset
                        //auto pledgeasset_ = (double) sharetoken_*pledgeratio;
                        auto pledgeasset_ = allminingasset.amount - sharetoken; // 成
都链安 // 计算本次抵押资产数量(挖矿总资产数量 - 账户分发量)
                        // pledgeasset_ *= std::pow(10,4);
                        auto _pledgeasset = pledgeasset.amount + pledgeasset_; // 成
都链安 // 更新该用户的抵押资产
                        extended_asset _pledgeasset_(_pledgeasset,S(4,POST));
// pledgeasset
                        _pledgeasset_.contract = name;
                        auto _totalasset = totalasset.amount +
allminingasset.amount; // 成都链安 // 更新本次挖矿后的总奖励
                        extended_asset _totalasset_(_totalasset,S(4,POST));
// totalasset
```

```
                    _totalasset_.contract = name;
                    if(pledgeitr!=_pledges.end()){ // 成都链安 // 如果该地址已经有抵
押数据，则更新数据
                        _pledges.modify(pledgeitr,name,[&](auto& p){
                            p.pledgeasset = _pledgeasset_;
                            p.totalasset = _totalasset_;
                        });
                    }else{ // 成都链安 // 如果对应地址的抵押记录不存在，则新增记录
                        _pledges.emplace(name,[&](auto &p){
                            p.address = address;
                            p.pledgeasset = _pledgeasset_;
                            p.totalasset = _totalasset_;
                            p.addtime = current_time();
                        });
                    }
                    // 成都链安 // 调用 issue 函数向目标地址 address 发行代币
                    action(
                        permission_level{N(ipsecontract),N(active)},
                        N(ipsecontract),N(issue),
                        std::make_tuple(address,amount,std::string("send POST"))
                    ).send();
                    print(address," mining ",sharetoken,"POST\n");
                    // ----------------------------------------settle---------
----------------------------------------


                    uint64_t now = current_time();
                    if((now - 86400000000)>settletime){ //Update the
addworkforce=0  totalworkforce=addworkforce  settletime=current_time(); // 成都链安 //
如果当前时间戳距离上一次结算时间已超过 1 天，则更新系统挖矿信息表
                        auto st = supplytoken(); // 成都链安 // 调用 supplytoken 函
数获取当前最大可增发数量
                        extended_asset amount(st,S(4,POST));
                        amount.contract = name;
                        _globalvars.modify(globalvars,name,[&](auto& p){ // 成都
链安 // 更新数据
                            p.addworkforce = force;
                            p.totalworkforce = addworkforce >=
10000000?addworkforce:10000000;
                            p.settletime = current_time();
                            p.balance = amount;
                        });
                    }else{ // addworkforce += force;
                        _globalvars.modify(globalvars,name,[&](auto& p){ // 成都链
安 // 如果未超过 1 天，则更新新增算力
                            p.addworkforce = addworkforce+force; // 成都链安 // 更
```

新新增算力

```
                        });
                    }
                }else{ // 成都链安 // 如果对应的挖矿信息表不存在，则初始化挖矿参数
                    auto st = supplytoken(); // 成都链安 // 调用 supplytoken 函数计算
当前可增发代币数量
                    extended_asset amount(st,S(4,POST));
                    amount.contract = name;
                    _globalvars.emplace(name,[&](auto &p){
                        p.key = 0;
                        p.totalworkforce = 500000000;
                        p.settletime = 1543143890000000;
                        p.increasetime = 1860310078000000;
                        p.balance = amount;
                        p.addworkforce  = 100;
                    });
                    print("add global vars success");
                }
            }else{
                print("wrong params");
            }
        }else{
            print("the pledge verify failed!");
        }
    }
};

// /// @abi action
// void findglobalva(uint64_t id){
//     auto& globalvar = _globalvars.get(0,"global vars doesn't exist");
//     print(globalvar.totalworkforce," ",globalvar.addworkforce,"
",globalvar.settletime," ",globalvar.balance.amount);
// };
// 成都链安 // 计算算力
uint64_t compute_workforce(uint8_t category,uint64_t size){
    const auto value = file.at(category);
    return size * value;
};

/// @abi action
// 成都链安 // 添加评议
void addreview(account_name address,account_name selfaddress){
    eosio_assert(is_account(address),"address account does not exist");
    account_name name = eosio::string_to_name("ipsecontract");
    auto blacklist_iterator = _blacklists.find(address);
    if(blacklist_iterator!=_blacklists.end()){ // 成都链安 // 如果被评议地址处于黑名
```

单中，则直接结束调用

```
            print("the address you review was in the blacklist");
        }else{
            auto blacklist_iterator_ = _blacklists.find(selfaddress);
            if(blacklist_iterator_!=_blacklists.end()){ // 成都链安 // 当 selfaddress 地址处于黑名单中
                print("you are in the blacklist!");
            }else{
                require_auth(selfaddress); // 成都链安 // 调用权限检查
                auto itr = _reviews.find(address);
                if(itr!=_reviews.end()){
                    _reviews.modify(itr,name,[&](auto& p){ // 成都链安 // 如果对应的被评议地址已经存在于被评议表中，则被评议次数+1
                        p.count++;
                    });
                    print("count review successfully \n");
                }else{
                    _reviews.emplace(name,[&](auto &p){ // 成都链安 // 如果对应的被评议地址不存在于被评议表中，则新增记录，被评议次数为 1
                        p.address = address;
                        p.count = 1;
                    });
                    print("add review successfully \n");
                }
                // 成都链安 // 更新评议者相关记录数据
                auto itrer = _reviewers.find(selfaddress);
                if(itrer!=_reviewers.end()){
                    _reviewers.modify(itrer,name,[&](auto& p){ // 成都链安 // 如果评议者已经存在于评议表中，则评议次数+1
                        p.count++;
                    });
                    print("count reviewer successfully \n");
                }else{
                    _reviewers.emplace(name,[&](auto &p){ // 成都链安 // 如果评议者不存在于评议表中，则新增记录，评议次数为 1
                        p.address = selfaddress;
                        p.count = 1;
                    });
                    print("add reviewer successfully \n");
                }
            }
        }

    };
```

```
    /// @abi action
    // 成都链安 // 查询指定账户的被评议次数
    void findreview(account_name address){
        auto& review = _reviews.get(address,"there is no address in the review
table");
        print(name{review.address}," ",int(review.count));
    };

    /// @abi action
    // 成都链安 // 查询指定账户的评议次数
    void findreviewer(account_name address){
        auto& reviewer = _reviewers.get(address,"there is no address in the reviewer
table");
        print(name{reviewer.address}," ",int(reviewer.count));
    };

    /// @abi action
    // 成都链安 // 根据索引查询被评议表中的对应信息
    void findreviewby(uint64_t count){
        account_name name = eosio::string_to_name("ipsecontract"); // 成都链安 // 该变量
未被使用，可删除
        auto countindex = _reviews.get_index<N(bycount)>();
        auto cou = countindex.find(count);
        for(;cou!=countindex.end();cou++){
            print("address:",cou->address," count:",cou->count);
        }
    };

    /// @abi action
    // 成都链安 // 根据索引查询评议表中的对应信息
    void getreviewerb(uint64_t count){
        account_name name = eosio::string_to_name("ipsecontract"); // 成都链安 // 该变量
未被使用，可删除
        auto countindex = _reviewers.get_index<N(bycount)>();
        auto cou = countindex.find(count);
        for(;cou!=countindex.end();cou++){
            print("address:",cou->address," count:",cou->count);
        }
    };

    /// @abi action
    // 成都链安 // 添加指定地址到黑名单
    void addblack(account_name address,uint64_t blocktime){
        account_name name = eosio::string_to_name("ipsecontract");
        require_auth(name); // 成都链安 // 调用权限检查
```

```
        if(blocktime>(current_time()/1000)){ // 成都链安 // 要求移除时间必须大于当前时间
            auto blacklist_iterator = _blacklists.find(address);
            if(blacklist_iterator!=_blacklists.end()){
                _blacklists.modify(blacklist_iterator,get_self(),[&](auto& p){ // 成都链安 // 如果指定地址已经在黑名单表中，则修改其黑名单状态移除时间
                    p.blocktime = blocktime;
                });
                print("update the address in blacklist scucess!");
            }else{
                _blacklists.emplace(name,[&](auto &p){ // 成都链安 // 如果指定地址不在黑名单表中，则新增记录并初始化数据
                    p.address = address;
                    p.blocktime = blocktime;
                });
                print("add blacklist element success!");
            }
        }else{
            print("blocktime must over now!");
        }
    }

    /// @abi action
    // 成都链安 // 仲裁
    void arbitrate(account_name address,uint8_t arbitrate_type,uint8_t result,uint64_t blocktime){ // arbitrate_type 0:review  1:reviewer  result 0:blacklist 1:free
        account_name name = eosio::string_to_name("ipsecontract");
        require_auth(name); // 成都链安 // 调用权限检查
        auto blacklist_iterator = _blacklists.find(address); // 成都链安 // 在黑名单中查找 address
        if(result==0){ // 成都链安 // 执行黑名单设置操作
            if(blacklist_iterator == _blacklists.end()){
                _blacklists.emplace(get_self(),[&](auto &p){ // 成都链安 // 对应地址不在黑名单中，则新增记录并初始化该地址的黑名单数据
                    p.address = address;
                    p.blocktime =  blocktime;
                });
                // if blocktime-now > 100 * 365*24*60*60*1000 confiscate the pledge pledgeasset
                if(blocktime > (current_time()/1000 + 3153600000000)){ // 成都链安 // 如果其移除时间距离当前时间大于 100 年，则回收其抵押的 POST
                    auto pledgeitr = _pledges.find(address);
                    if(pledgeitr!=_pledges.end()){
                        auto& pledge = _pledges.get(address);
```

```
                        extended_asset pledgeasset = pledge.pledgeasset;
                        extended_asset _pledgeasset_(0,S(4,POST));
                        _pledgeasset_.contract = name;
                        _pledges.modify(pledgeitr,name,[&](auto& p){
                            p.pledgeasset = _pledgeasset_;
                        });
                        print("confiscate POST from pledge table
success:",pledgeasset.amount,"\n");
                    }
                }
            }
            print("address:",N(address)," emplace into blacklist");
        }else{ // 成都链安 // 执行移除黑名单操作
            if(blacklist_iterator != _blacklists.end()){ // 成都链安 // 如果指定地址处于
黑名单中，则将其从黑名单中移除
                _blacklists.erase(blacklist_iterator);
            }else{
                print("address:",N(address)," do no exists in the blacklist");
            }
        }
        if(arbitrate_type==0){ // 成都链安 // 仲裁类型是 0
            auto review_iterator = _reviews.find(address);// delete the review
record
            // 成都链安 // 要求地址 address 处于被评议表中
            eosio_assert(review_iterator!=_reviews.end(),"the review address you
arbitrate is not in the review list!");
            _reviews.erase(review_iterator); // 成都链安 // 删除被评议者的记录信息
        }else{ // 成都链安 // 仲裁类型不是 0
            auto reviewer_iterator = _reviewers.find(address);// delete the reviewer
record
            eosio_assert(reviewer_iterator!=_reviewers.end(),"the reviewer address
you arbitrate is not in the reviewer list!");
            _reviewers.erase(reviewer_iterator); // 成都链安 // 删除该账户的评议记录信息
        }
    };
};

extern "C" {
    void apply(uint64_t receiver,uint64_t code,uint64_t action){
        ipsecontract thiscontract(receiver);
        // 成都链安 // 如果对应的合约是 ipsecontract、action 是 transfer，则执行本合约的
transfer 函数
        if((code==N(ipsecontract)) && (action==N(transfer))){
            execute_action(&thiscontract,&ipsecontract::transfer);
            return;
```

```
        }
        if(code!=receiver) return; // 成都链安 // 如果 code 与 receiver 不相同，则直接
return
        // 成都链安 // 本合约 API
        switch (action)
{EOSIO_API(ipsecontract,(create)(increase)(issue)(transfer)(findpledge)(addblacklist
)(findblacklis)(eraseblackli)(modifyblackl)(version)(reducepledge)(addlabel)(addrevi
ew)(findreview)(findreviewer)(findreviewby)(getreviewerb)(addblack)(arbitrate))};
        eosio_exit(0);
    }
}
```

成都链安
BEOSIN

官方网址

https://lianantech.com

电子邮箱

vaas@lianantech.com

微信公众号