xcertauth 接口文档

功能介绍:

xcertauth 是以多重门限签名为基础的证书&认证的c++实现，提供单签名，多签名合并 和检验等功能。主要基础是Schnorr门限签名算法。

基本术语:

Schnorri：一种安全多重门限签名算法，特点是数据量适中而可快速检验。 参考

xvqcert_t: 类似CA证书的封装,包含要证明的Hash值, 以及来自2类节点身份的签名和认证信息：

Validator: 共识检验节点, 每个节点有唯一的虚拟xvip2_t地址

Auditor: 共识审计节点，每个节点有唯一的虚拟xvip2_t地址

xvqcert_t的时钟高度和选举块高度决定了哪些Validator&Auditor 是合法的检验者

xvblock_t: 对任何block的抽象封装，每个xvblock都包含input/output/header 以及 xvqcert_t证明

xvnodesrv_t： 每一轮选举出来的节点集合

unit chain：每个用户有一个唯一对应的账号地址(account address)，也有自己的一条Unit 链。

Unit 链是由Unit Block链接而成

基本签名流程:

1. 确定xvqcert_t的时钟高度和选举块高度，查询xvnodesrv_t 获得合法的Validator&Auditor 节点集
2. 共识过程每个Validator节点对共识内容进行裁决，并调用do_sign()对内容签名并送回给签名聚合者(Leader)
3. 签名聚合者(Leader) 使用verify_sign()检验单个Validator节点的签名
4. 签名聚合者(Leader) 使用merge_muti_sign对把来自Validator的所有有效的单签名聚合成最终的Schnorri多重签名
5. 类似上面#2，#3，#4的过程，审计节点进行对共识内容和validtor的多重签名进行检验，并产生Auditor节点群的聚合的Schnorri多重签名
6. 组合 Validtor 和 Auditor的2个多重签名，为xvqcert_t的最终证明。

代码：

1. xxx/src/xtopcom/xcertauth/为主入口代码,跨平台编译在CMakeLists.txt, XCode项目文件为xcertauth.xcodeproj
2. xxx/src/xtopcom/xmutisig/ 为schnorri算法实现,跨平台编译在CMakeLists.txt, XCode项目文件为xmutisig.xcodeproj
3. xxx/src/xtopcom/xbase/ 为基础结构&基础API定义所在
4. 简单的sample code 位于 xxx/src/xtopcom/xcertauth/test/basic

xcertauth对外接口：

```
//dependon xvnodesrv_t that manage nodes from election
class xauthcontext_t : public base::xvcertauth_t
{
    public:
       //use global instance even for case of simulation of mutiple nodes
      static base::xvcertauth_t&  instance(base::xvnodesrv_t & node_service);
}
```

```
//Certificate-Authority
  class xvcertauth_t : public xobject_t //CA system
  {
          friend class xvheader_t;
      public:
          static  const std::string   name(){ return std::string("xvcertauth");}
          virtual std::string         get_obj_name() const override {return 
name();}
      protected:
          xvcertauth_t();
          virtual ~xvcertauth_t();
      private:
          xvcertauth_t(const xvcertauth_t &);
          xvcertauth_t & operator = (const xvcertauth_t &);

      public:
          virtual const std::string   get_signer(const xvip2_t & signer) = 0;
//query account address of xvip2_t
          //all returned information build into a xvip_t structure
          virtual xvip_t              get_validator_addr(const std::string & 
account_addr) = 0; //mapping account to target group
          virtual bool                verify_validator_addr(const base::xvblock_t 
* test_for_block) = 0;//verify validator and account
```

```
        virtual bool                verify_validator_addr(const std::string &
for_account,const base::xvqcert_t * for_cert) = 0;//verify validator and account

      public: //returned_errcode parameter carry detail error if verify_muti_sign
fail(return false)
        //random_seed allow pass a customzied random seed to provide unique
signature,it ask xvcertauth_t generate one if it is 0
        //signature by owner ' private-key
        virtual const std::string   do_sign(const xvip2_t & signer,const
base::xvqcert_t * sign_for_cert,const uint64_t random_seed)   = 0;//random_seed is
optional
        virtual const std::string   do_sign(const xvip2_t & signer,const
base::xvblock_t * sign_for_block,const uint64_t random_seed)= 0;//random_seed is
optional

        virtual enum_vcert_auth_result   verify_sign(const xvip2_t &
signer,const xvqcert_t * test_for_cert,const std::string & block_account)  = 0;
        virtual enum_vcert_auth_result   verify_sign(const xvip2_t &
signer,const xvblock_t * test_for_block) = 0;

      public:
        //merge multiple single-signature into threshold signature,and return a
merged signature
        virtual const std::string   merge_muti_sign(const std::vector<xvip2_t> &
muti_nodes,const std::vector<std::string> & muti_signatures,const xvqcert_t *
for_cert) = 0;
        virtual const std::string   merge_muti_sign(const
std::map<xvip2_t,std::string,xvip2_compare> & muti_nodes_signatures,const
xvqcert_t * for_cert) = 0;
        virtual const std::string   merge_muti_sign(const
std::map<xvip2_t,std::string,xvip2_compare> & muti_nodes_signatures,const
xvblock_t * for_block) = 0;

      public://returned_errcode parameter carry detail error if verify_muti_sign
fail(return false)
        //note:just verify multi-sign of group is ok for 'sign_hash', but not
check whether the sign_hash is good or not
        virtual enum_vcert_auth_result   verify_muti_sign(const xvqcert_t *
test_for_cert,const std::string & block_account) = 0;

        //note:check from ground: generate/check vbody'hash->  generate/check
vheader'hash -> generate/check vqcert'sign-hash-> finally verify multi-signature
of group. for safety please check threshold first to see it was ready
        virtual enum_vcert_auth_result   verify_muti_sign(const xvblock_t *
test_for_block) = 0;
  };
```

主要功能API:

```
virtual const std::string   do_sign(const xvip2_t & signer,const
base::xvqcert_t * sign_for_cert,const uint64_t random_seed)//random_seed is
optional
virtual const std::string   do_sign(const xvip2_t & signer,const
base::xvblock_t * sign_for_block,const uint64_t random_seed)//random_seed is
optional
```

Signer(节点）对目标的xvqcert_t证书的内容进行签名，返回签名数据

```
//returned_errcode parameter carry detail error if verify_muti_sign
fail(return false)
//random_seed allow pass a customzied random seed to provide unique
signature,it ask xvcertauth_t generate one if it is 0
//signature by owner ' private-key

virtual enum_vcert_auth_result   verify_sign(const xvip2_t & signer,const
xvqcert_t * test_for_cert,const std::string & block_account);
virtual enum_vcert_auth_result   verify_sign(const xvip2_t & signer,const
xvblock_t * test_for_block);
```

检验目标的xvqcert_t证书携带的单个签名是否来自signer的有效签名

```
//merge multiple single-signature into threshold signature,and return a merged
signature
virtual const std::string   merge_muti_sign(const std::vector<xvip2_t> &
muti_nodes,const std::vector<std::string> & muti_signatures,const xvqcert_t *
for_cert) = 0;

virtual const std::string   merge_muti_sign(const
std::map<xvip2_t,std::string,xvip2_compare> & muti_nodes_signatures,const
xvqcert_t * for_cert) = 0;

virtual const std::string   merge_muti_sign(const
std::map<xvip2_t,std::string,xvip2_compare> & muti_nodes_signatures,const
xvblock_t * for_block) = 0;
```

为xvqcert_t 聚合muti_nodes_signatures 成 Schnorr 的门限多重签名

```
//returned_errcode parameter carry detail error if verify_muti_sign
fail(return false)
//note:just verify multi-sign of group is ok for 'sign_hash', but not check
whether the sign_hash is good or not

virtual enum_vcert_auth_result   verify_muti_sign(const xvqcert_t *
test_for_cert,const std::string & block_account) = 0;

//note:check from ground: generate/check vbody'hash->  generate/check
vheader'hash -> generate/check vqcert'sign-hash-> finally verify multi-
signature of group. for safety please check threshold first to see it was
ready

virtual enum_vcert_auth_result   verify_muti_sign(const xvblock_t *
test_for_block) = 0;
```

检验xvblock_t/xvqcert_t 是否包含证明来自合法选举节点集的，有效的多重签名。先检查检验节点群(Validators)的门限签名，再检查审计节点群(Auditors)的门限签名

模块API:

```
//use global instance even for case of simulation of mutiple nodes
static base::xvcertauth_t&  instance(base::xvnodesrv_t & node_service);
```