Trias 隐私交易应用设计

区块链utxo模式下的交易流程

基于utxo工作模式下的加密货币区块链系统的交易流程可以简单划分为以下两个步骤：

1. 交易发起方结合需求产生交易

   假设用户A需要对用户B发起一笔交易，用户A计划使用 U_a 进行支付，并产生支付给用户B的 U_b ,结余 U_r，使用的手续费为 F，用户A将对如上数据组成交易数据并在关键部分签名之后进行广播，完成交易的发起过程。

$$P = [sign(U_a), U_a, U_b, U_r, F] \tag{1}$$

2. 网络上的矿工验证、记录并执行交易

   矿工接收到广播的交易之后，首先需要进行如下验证：

   - U_a 是否是一个有效的utxo，即必须是一笔正常的未花费支出
   - Sign(U_a)是否是有效签名，确定付款人是否拥有对应支配权
   - U_a的余额等于U_b,U_r,F 的和，即

$$Balance_{u_a} = Balance_{u_b} + Balance_{u_r} + F \tag{2}$$

在完成如上的3个校验之后，矿工才会记录并执行此笔交易。综上所述流程可以发现，对矿工来说，其不关心**发款人是谁，收款人是谁，付款多少**，只需要交易信息中的检验条件通过，就视为一个有效的交易。如此分析可知，通过此种交易模式，我们可以设计一种隐私交易模式，屏蔽交易中的多个明文信息，同时提供能**让矿工相信交易有效的证明数据**。

Trias 隐私交易设计

隐私交易需求

矿工为了能够直白的验证对应的交易是否合法，需要交易发起人提供明文的U_a, U_b, U_c,而其中则具体包含了账户的明文地址信息$Addr_a, Addr_b$，金额信息 $Balance_{u_a}, Balance_{u_b}, Balance_{u_c}$。

在高级业务场景需求中，trias为其提供了更高级别的隐私交易功能，实现了：

- 发起人，收款人账户地址隐藏
- 交易中除手续费之外的资金信息隐藏

隐私交易设计

经过前面的交易流程分析，矿工的校验，记录和执行操作在数据层面上是不需要如上的明文信息的，其需要的只是经过证明后交易合法的证据。在一般的设计实现中，为了让矿工获取交易合法的证据，打款人把所有信息明文显示，矿工基于此信息进行后续的合法性判断。过程化描述如下：

$$U_i = [Addr_x, Balance_{u_i}] \\ Verify(sign(U_a), U_a, U_b, U_r, F) \tag{3}$$

为了实现对账户地址和资金的全方面的信息隐藏，trias采用隐私地址的方案实现地址隐藏，采用零知识证明实现资金信息隐藏。

账户地址隐藏-隐藏地址

假设账户A需要向账户B支付5个token，账户B有两套ECC算法框架下的公私钥对（M=m*G*,*N*=*n*G,M,N为公钥，m,n为私钥，G为指定的ECC椭圆曲线），常规的转账产生的UTXO为 $U_b = [Addr_b, 5]$, 其表示向账户B的地址转账5个token。

```
//core/shield.go

//创建隐私地址
//根据输入目标用户地址，产生对应用户地址的隐私地址
func CreateShieldAddr(addr string) ([]byte, []byte ) {
    //gkeyA := acc.GkeyA
    //gkeyB := acc.GkeyB
    pubk:=GetPubk4Addr(addr) //根据目标地址解析获得对应公钥信息
    if pubk==nil{
        return nil,nil
    }
    A_X:=pubk.X
    A_Y:=pubk.Y
    B_X:=pubk.X
    B_Y:=pubk.Y
    randomkey, _ := ecdsa.GenerateKey(curve,
strings.NewReader(lib.GenerateRstring(45)))


    P, _ := curve.ScalarMult(A_X, A_Y, randomkey.D.Bytes()) //Mr

    x, y := curve.ScalarBaseMult(P.Bytes()) //(Mr)G

    P, _ = curve.Add(x, y, B_X, B_Y) //(Mr)G+N

    pubkey :=
append(lib.PubkeyPad(randomkey.PublicKey.X.Bytes()),lib.PubkeyPad(randomkey.PublicKe
y.Y.Bytes())...)
    return P.Bytes(),pubkey  // 返回隐私地址数据，随机数R
}

```

- 发起隐藏地址交易

  当采用隐私地址技术时，则会通过目标账户的公钥和随机一次性秘钥生成临时地址进行转账，如此以达到对账户地址的隐私保护过程。过程如下，其最终生成的UTXO为$U_{b'}$。

$$\begin{cases} \text{随机生成的数字：} R = r * G \\ \text{账户} B \text{的公钥} M, N \end{cases} \tag{4}$$
$$Shield_{addr_b} = sha256(Mr) * G + N$$
$$U_{b'} = [Shield_{addr_b}, R, 5]$$

```go
//core/shield.go

//验证隐私地址
//根据输入的隐私地址byte和随机数R，验证对应隐私地址是否为当前输入账户拥有
func Verify_shield(acc *Account, shieldaddr []byte, shieldpKey []byte) bool {
    gkeyA := acc.GkeyA
    gkeyB := acc.GkeyB

    P := new(big.Int).SetBytes(shieldaddr)

    R_x := new(big.Int).SetBytes(shieldpKey[:32])
    R_y := new(big.Int).SetBytes(shieldpKey[32:])

    p, _ := curve.ScalarMult(R_x, R_y, gkeyA.PrivateKey.D.Bytes()) //mR

    x, y := curve.ScalarBaseMult(p.Bytes()) //(mR)G

    p, _ = curve.Add(x, y, gkeyB.PublicKey.X, gkeyB.PublicKey.Y) //(mR)G+N

    if P.Cmp(p) == 0 { //判断P==p,若相等则验证通过，输入的隐私地址为当前账户拥有
        return true
    }
    return false
}
```

- 收款并支付

  账户B遍历所有交易，基于自身私钥和$U_{i'}$中的随机数R，计算验证判断是否自身收款项，若以下判断为真，则为自身收款，过程如下：

$$P = sha256(mR) * G + N \tag{5}$$
$$P == Shield_{addr_b}$$

  当账户B需要使用这笔采用了隐私地址计算转账的资金时，需要通过计算恢复出一次性地址的私钥并进行签名，如此进行支付，过程如下：

$$P = sha256(mR) * G + N = sha256(mR) * G + n * G = (sha256(mR) + n) * G \tag{6}$$
$$\text{Private key: } sha256(mR) + n$$

```go
//core/shield.go

//计算获得隐私地址对应私钥
//根据隐私地址，随机数R，账号私钥，计算出隐私地址对应私钥，后续使用此私钥对隐私地址对应utxo进行支付签名
func Getprivkey(acc *Account, shieldpKey []byte) *ecdsa.PrivateKey {
    gkeyA := acc.GkeyA
```

```
    gkeyB := acc.GkeyB

    R_x := new(big.Int).SetBytes(shieldpKey[:32])
    R_y := new(big.Int).SetBytes(shieldpKey[32:])

    x, _ := curve.ScalarMult(R_x, R_y, gkeyA.PrivateKey.D.Bytes()) //mR
    x = x.Add(x, gkeyB.PrivateKey.D)                               //(mR+n)

    priv := new(ecdsa.PrivateKey)
    priv.PublicKey.Curve = curve
    priv.D = x
    priv.PublicKey.X, priv.PublicKey.Y = curve.ScalarBaseMult(x.Bytes()) //(mR+n)G
    return priv //返回私钥信息
}
```

- 矿工验证交易

  在接收到经过隐私地址处理的交易后，矿工首先需要对其进行合法性检测，具体内容在第一章节有过描述。经过分析可以发现，矿工对此类交易是透明感知的，其校验过程和传统的校验手段一致，无需引入额外的计算验证。

交易金额隐藏-零知识证明

通过上一节的内容，trias实现了交易双方地址信息的隐藏；针对交易金额的隐藏，trias采用了零知识证明计算，验证交易中的金额符合 $Balance_{u_a} = Balance_{u_b} + Balance_{u_r} + F$。

现假设一笔自账户A到账户B的交易：

$$\begin{aligned} U_i &= [Addr_x, Balance_{u_i}] \\ T_i &= [sign(U_a), U_a, U_b, U_r, F] \end{aligned} \tag{7}$$

其中关于金额的明文变量有：$Balance_{u_a}, Balance_{u_b}, Balance_{u_c}$ 数值关系为：
$Balance_{u_a} = Balance_{u_b} + Balance_{u_r} + F$。

当针对采用了零知识证明进行金额隐藏的UTXO结构如下：

$$U_i' = [Addr_x, E_x(Balance_{u_i'}), commit_i, r_i] \tag{8}$$

其中定义E_x 为采用对应用户公钥加密的函数，r 为每一个隐私 U_i' 都随机生成的一个随机数，commit 为一个支付承若，本质上一个包含了金额数据的哈希处理，如下：

$$commit_i = hash(Addr_x, Balance_{u_i'}, r_i) \tag{9}$$

当发起隐私支付时，其过程如下：

$$\begin{aligned} U_a' &= [Addr_a, E_a(Balance_{u_a'}), commit_a, r_a] \\ U_b' &= [Addr_b, E_b(Balance_{u_c'}), commit_b, r_c] \\ U_c' &= [Addr_c, E_c(Balance_{u_c'}), commit_c, r_c] \\ P &= Prove(Addr_a, Addr_b, Addr_c, Balance_{u_a'}, Balance_{u_b'}, Balance_{u_c'}, r_a, r_b, r_c, F, keypairs) \\ T_i' &= [sign(U_{a'}), U_{a'}, U_{b'}, U_{c'}, F, P] \end{aligned} \tag{10}$$

上述T'即为采用零知识证明后的金额隐私交易数据，此数据向公链公开，并需要使得矿工节点（验证节点）的验证并记录,其中keypairs为zkSNARK算法初始化生成的秘钥对序列，P为支付发起方生成的证明，其表明对应支付的中支付金额符合如下关系表达式：

$$Balance_{u'_a} = Balance_{u'_b} + Balance_{u'_r} + F \tag{11}$$

矿工在接收到到广播的金额隐藏交易时，处理过程如下：

$$Verify(U_{a'}, U_{b'}, U_{c'}, F, P, keypairs) \tag{12}$$

当上述表达式验证为true，矿工即可以认为对应交易中的金额符合表达式：
$Balance_{u'_a} = Balance_{u'_b} + Balance_{u'_r} + F$。

再当完成对应输入U'_x 和 $sign(U'_a)$的有效性检查后，各个验证节点即认可对应的交易合法，最后全网达成共识后记录到整个区块链系统上去。

```
对以上的零知识证明应用，可以归纳成如下过程：
A已知：
 - addr_a,amount_a,r_a,addr_b,amount_b,r_b,amount_c,r_b,f
 - H1=hash(addr_a,amount_a,r_a),H2=hash(addr_b,amount_b,r_b),H3=hash(addr_c,amount_c,r_c)
 - amount_a=amount_b+amount_c+f

 B已知：
  - addr_a,r_a,addr_b,r_b,addr_c,r_c,f
  - H1，H2,H3

 现需要A在目前A,B已知数据的基础和限制下向B证明：
  - A 拥有amount_a,amount_b,amount_c三个数
  - amount_a=amount_b+amount_c+f


 零知识证明数学描述：
 定义：
 a_i=md5(addr_i)
 r_i=[16]byte (random)

 R_1=a_1+r_1+m_1
 R_2=a_2+r_2+m_2
 R_3=a_3+r_3+m_3
 H_1=hash(R_1)
 H_2=hash(R_2)
 H_3=hash(R_3)
 m_1=m_2+m_3+f (手续费)
 => R_1=R_2+R_3+(a_1+r_1-a_2-r_2-a_3-r_3+f) (括号中的为公开数据，定义为X=a_1+r_1-a_2-r_2-a_3-r_3+f)
 R_1=R_2+R_3+X

 综上：
 证明端输入为：a1,r1,m1,a2,r2,m2,a3,r3,m3,[f]
 计算获得R_1,R_2,R_3,X,H1,H2,H3,并基于零知识证明获得P
 公开：X,H1,H2,H3,P
```

```
 验证端基于: a1,r1,a2,r2,a3,r3,[f],H1,H2,H3,P
 计算获得X
 最后基于: X,H1,H2,H3,P 完成零知识证明验证

 基于zksnark的原型demo: https://github.com/yixuanzi/lightning_circuit

 update:
 为支持多重输入和多重输出，在trias的utxo交易模式下，重新定义数学问题如下:
 设定当前零知识证明支持M个输入，N个输出
 定义:
 a_i=md5(addr_i) （公开）
 r_i=[16]byte (random)  （公开）
 R_i=a_i+r_i+m_i  （保密）
 H_i=hash(R_i) （公开）
 m_i_1+m_i_2+...+m_i_M=m_i_o+m_o_2+...+m_o_N+f
 ar_i=a_i+r_i

 =>R_i_1+R_i_2+...+R_i+M=R_o_1+R_o_2+...+R_o_N+ar_i_1+ar_i_2+...+ar_i_M-ar_o_1-
ar_o_2...-ar_o_N+f
 设 X=ar_i_1 + ar_i_2 +...+ ar_i_M - ar_o_1 - ar_o_2...- ar_o_N + f
 则 当X>0:
          R_i_1+R_i_2+...+R_i+M=R_o_1+R_o_2+...+R_o_N+X
 则 当X<0:
          R_i_1+R_i_2+...+R_i+M -X =R_o_1+R_o_2+...+R_o_N (转移X，使减法变加法)

 证明生产端:
 由R_i_1,R_i_2,....R_i_M,R_o_1,R_o_2...,R_o_N,X
 生成并公开: H_i_1,H_i_2,...H_i_M,H_o_1,H_o_2,...,H_o_N,X,P

 验证端:
    Virity(H_i_1,H_i_2,...H_i_M,H_o_1,H_o_2,...,H_o_N,X,P)

 ======================
 当交易中一方是金额隐藏保护，一方是开放的，定义其金额平衡表达式为 m_1+m_2+..m_N=M+f
 基于上述假设定义，零知识证明的计算表达式可转化为:
 R_1+R2+...+R_N=M+ar_1+ar_2+...+ar_N+f (当等式两边中关于隐藏和开放的数据交换时，f值需要减去)

 定义: X=M+ar_1+ar_2+...+ar_N+f
则证明生成端为:
R_1,R_2,..R_N,X =>P,H_1,H_2,...H_N,X

验证端:
    Virity(H_1,H_2,...H_N,X,P)
 ====================================
 当实际输入输出端未达到计算约定的M，N个时，使用空白加密承诺进行等式平衡计算，如下:
 a_blank=0
 r_blank=0
 m_blank=0
 ar_blank=0
 R_blank=a_blank+r_blank+m_blank=0
 H_blank=hash(R_blank)

```

以上，可以在当前传统区块链架构的基础上除了增加零知识证明相关的计算资源外，不消耗其他资源，不改变交易检验架构的实现了多重输入和多重输出。