混淆证明

Background - ElGamal encryption

在实现中使用的是ECC椭圆曲线，结果也正常

- Setup: Group $\mathcal{G}$ of prime order q with generator g
- Public key: $\mathrm{pk} = y = g^x$
- Encryption: $\mathcal{E}_{pk}(m; r) = (g^r, y^r m)$
- Decryption: $\mathcal{D}_x(u, v) = vu^{-x}$
- Homomorphic:

$$\mathcal{E}_{pk}(m; r) \times \mathcal{E}_{pk}(M; R) = \mathcal{E}_{pk}(mM; r + R)$$

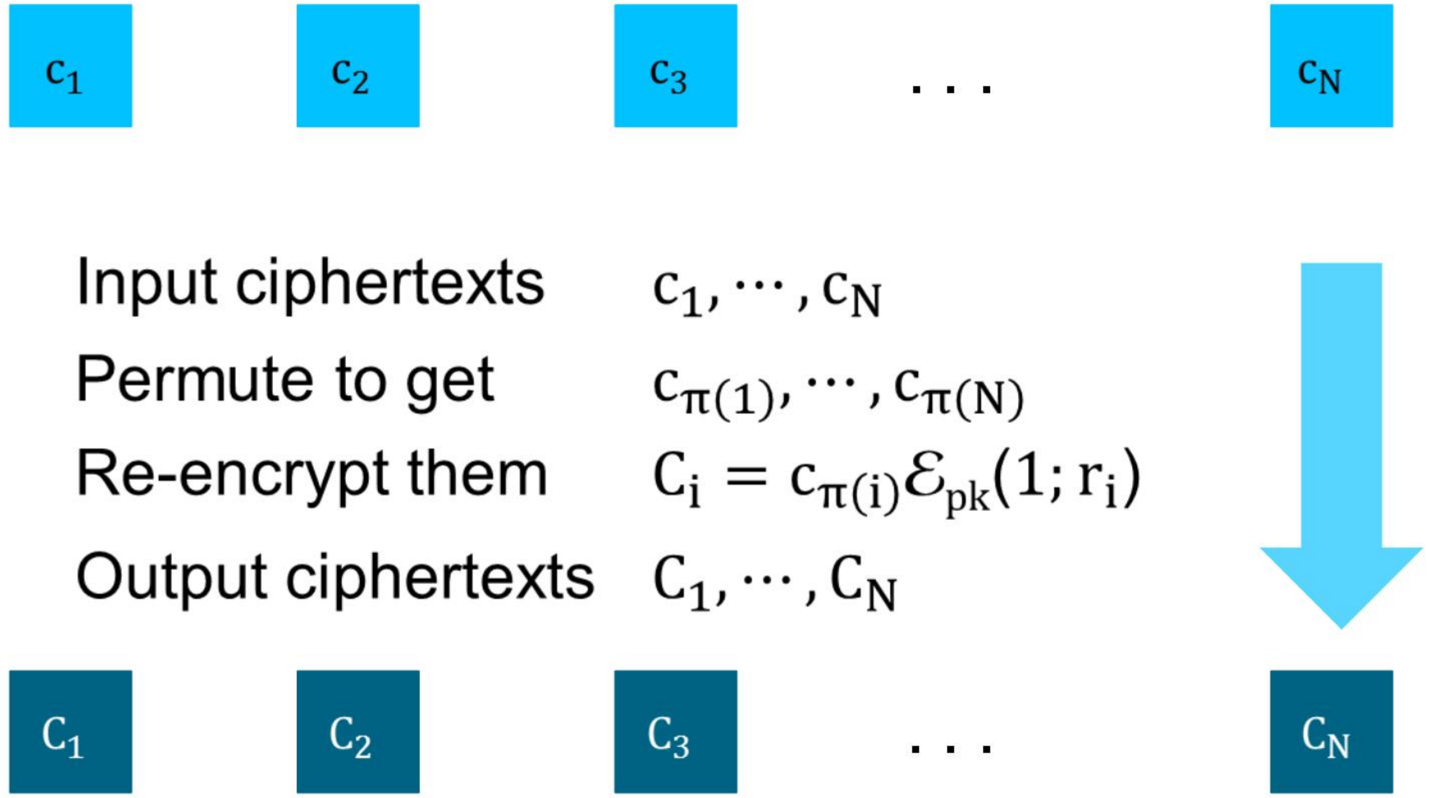
- Re-rencryption:

$$\mathcal{E}_{pk}(m; r) \times \mathcal{E}_{pk}(1; R) = \mathcal{E}_{pk}(m; r + R)$$

此处加密方案直接用的m*h^r,因为之后的流程主要用到这个同态性，没有涉及到解密操作，把前一半g^r忽略了。

方案出自：EUROCRYPT 2012：Efficient Zero-Knowledge Argument for Correctness of a Shuffle

总体思路



此处c代表Pederson承诺，实验中使用的是：对单个数值和单个随机数承诺(作为输入输出)：

Com(v, r) = $g^{v}h^{r}$

在计算过程中还会生成一些承诺作为中间参数，其形式为：

对向量和一个随机数承诺：

$\mathrm{Com}(\vec{v}, r) = g^{v1}g^{v2} \ldots g^{r}$

对向量和多个随机数承诺：

$\mathrm{Com}(\vec{v}, \vec{r}) = g^{v1}h^{r1}, g^{v2}h^{r2} \ldots g^{vn}h^{rn}$

π代表输入的更新序列：例如输入4个承诺C1, C2, C3, C4, π为[4,3,2,1]

则混淆后的承诺为 C4', C3', C2', C1' 加了'因为每个承诺的随机数有变化

算法流程

Common reference string: pk, ck.
Statement: $\boldsymbol{C}, \boldsymbol{C}' \in \mathbb{H}^N$ with $N = mn$.
Prover's witness: $\pi \in \Sigma_N$ and $\boldsymbol{\rho} \in \mathbb{Z}_q^N$ such that $\boldsymbol{C}' = \mathcal{E}_{pk}(\mathbf{1}; \boldsymbol{\rho})\boldsymbol{C}_\pi$.
Initial message: Pick $\boldsymbol{r} \leftarrow \mathbb{Z}_q^m$, set $\boldsymbol{a} = \{\pi(i)\}_{i=1}^N$ and compute $\boldsymbol{c}_A = \mathrm{com}_{ck}(\boldsymbol{a}; \boldsymbol{r})$.
Send: $\boldsymbol{c}_A$
Challenge: $x \leftarrow \mathbb{Z}_q^*$.
Answer Pick $\boldsymbol{s} \in \mathbb{Z}_q^m$, set $\boldsymbol{b} = \{x^{\pi(i)}\}_{i=1}^N$ and compute $\boldsymbol{c}_B = \mathrm{com}_{ck}(\boldsymbol{b}; \boldsymbol{s})$.
Send: $\boldsymbol{c}_B$
Challenge: $y, z \leftarrow \mathbb{Z}_q^*$.
Answer: Define $\boldsymbol{c}_{-z} = \mathrm{com}_{ck}(-z, \ldots, -z; \mathbf{0})$ and $\boldsymbol{c}_D = \boldsymbol{c}_A^y \boldsymbol{c}_B$. Compute $\boldsymbol{d} = y\boldsymbol{a} + \boldsymbol{b}$, and $\boldsymbol{t} = y\boldsymbol{r} + \boldsymbol{s}$. Engage in a product argument as described in Sect. 5 of openings $d_1 - z, \ldots, d_N - z$ and $\boldsymbol{t}$ such that

① $$\boldsymbol{c}_D \boldsymbol{c}_{-z} = \mathrm{com}_{ck}(\boldsymbol{d} - \boldsymbol{z}; \boldsymbol{t}) \quad \text{and} \quad \prod_{i=1}^N (d_i - z) = \prod_{i=1}^N (yi + x^i - z).$$

Compute $\rho = -\boldsymbol{\rho} \cdot \boldsymbol{b}$ and set $\boldsymbol{x} = (x, x^2, \ldots, x^N)^T$. Engage in a multi-exponentiation argument as described in Sect. 4 of $\boldsymbol{b}$, $\boldsymbol{s}$ and ρ such that

② $$\boldsymbol{C}^{\boldsymbol{x}} = \mathcal{E}_{pk}(1; \rho)\boldsymbol{C}'^{\boldsymbol{b}} \quad \text{and} \quad \boldsymbol{c}_B = \mathrm{com}_{ck}(\boldsymbol{b}; \boldsymbol{s})$$

The two arguments can be run in parallel. Furthermore, the multi-exponentiation argument can be started in round 3 after the computation of the commitments $\boldsymbol{c}_B$.
Verification: The verifier checks $\boldsymbol{c}_A, \boldsymbol{c}_B \in \mathbb{G}^m$ and computes $\boldsymbol{c}_{-z}, \boldsymbol{c}_D$ as described above and computes $\prod_{i=1}^N (yi + x^i - z)$ and $\boldsymbol{C}^{\boldsymbol{x}}$. The verifier accepts if the product and multi-exponentiation arguments both are valid.

Theorem 5 (Full paper). *The protocol is a public coin perfect SHVZK argument of knowledge of* $\pi \in \Sigma_N$ *and* $\boldsymbol{\rho} \in \mathbb{Z}_q^N$ *such that* $\boldsymbol{C}' = \mathcal{E}_{pk}(\mathbf{1}; \boldsymbol{\rho})\boldsymbol{C}_\pi$.

改成非交互式时，用的
x = Hash(g,h,$\overrightarrow{CA}$)
y = Hash(g,h,$\overrightarrow{CA}$,$\overrightarrow{CB}$)
Z = Hash(g,h,$\overrightarrow{CA}$,$\overrightarrow{CB}$,y)
正确与否还需验证

此处乘积证明用到两个其他的sigma协议，文章给出了参考文献

此处多项式幂的证明也在这个文章里，后边有说

① 乘积证明

乘积证明Prover要证明CA1,CA2…中的value a1,a2… 相乘为x
乘积证明分为两个步骤：1Prover证明其知道CA1,CA2,CA3…的witness； 2Prover证明乘积

第一步：证明知道承诺里面的数值(此协议简称为KCO)

Consider a set of commitments $c_1, \ldots, c_m$ given by $c_i = \mathrm{com}(\boldsymbol{x}_i; r_i)$. We will now give a 3-move public coin perfect SHVZK argument of knowledge of the committed vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m \in \mathbb{Z}_p^n$.

P→ V: The prover on input of a commitment key ck, a statement consisting of m commitments $c_1, \ldots, c_m$ and openings of the commitments $\{(\boldsymbol{x}_i, r_i)\}_{i=1}^m$ chooses $\boldsymbol{x}_0 \leftarrow \mathbb{Z}_p^n, r_0 \leftarrow \mathbb{Z}_p$ and sends $c_0 = \mathrm{com}_{ck}(\boldsymbol{x}_0; r_0)$ to the verifier.

P← V: The verifier sends the prover a random challenge $e \leftarrow \mathbb{Z}_p$.

P→ V: The prover answers the challenge with $\boldsymbol{z}, s$ computed as $\boldsymbol{z} = \sum_{i=0}^m e^i \boldsymbol{x}_i$ and $s = \sum_{i=0}^m e^i r_i$.

V: The verifier accepts the argument if $\prod_{i=0}^m c_i^{e^i} = \mathrm{com}_{ck}(\boldsymbol{z}; s)$.

此处challenge e改成非交互的时候实现的时候用的是Hash(g, h, $\vec{x}$, $\vec{r}$, $\vec{c}$,)正确与否还需验证

出自 Linear Algebra with Sub-linear Zero-Knowledge Arguments) CRYPTO 2009 的附录A

1 乘积证明

第二步：证明乘积(此文件中简称为ProdctProof)

proof-of-product(X, Y, Z)

Inputs: $X = g^x \odot h^{r_X}$, $Y = g^y \odot h^{r_Y}$, and $Z = g^{x \cdot y} \odot h^{r_Z}$.
$\mathcal{P}$ knows x, y, r_X, r_Y, and r_Z.

1. $\mathcal{P}$ picks $b_1, \ldots, b_5 \xleftarrow{\$} \{1, \ldots, q_{\mathcal{G}}\}$ and sends

$$\alpha \leftarrow g^{b_1} \odot h^{b_2} \quad \beta \leftarrow g^{b_3} \odot h^{b_4} \quad \delta \leftarrow X^{b_3} \odot h^{b_5}$$

2. $\mathcal{V}$ sends a challenge $c \xleftarrow{\$} \{1, \ldots, q_{\mathcal{G}}\}$
3. $\mathcal{P}$ sends

$$z_1 \leftarrow b_1 + c \cdot x \quad z_2 \leftarrow b_2 + c \cdot r_x \quad z_3 \leftarrow b_3 + c \cdot y$$
$$z_4 \leftarrow b_4 + c \cdot r_y \quad z_5 \leftarrow b_5 + c(r_z - r_x y)$$

4. $\mathcal{V}$ checks that

$$\alpha \odot X^c \stackrel{?}{=} g^{z_1} \odot h^{z_2} \tag{7}$$
$$\beta \odot Y^c \stackrel{?}{=} g^{z_3} \odot h^{z_4} \tag{8}$$
$$\delta \odot Z^c \stackrel{?}{=} X^{z_3} \odot h^{z_5} \tag{9}$$

参考的算法是对两个数乘积的证明，
此处实现时分别对a1, a2, … an 中a1*a2, a2*a3, a3*a4 … an-1*an, an*a1 的乘积进行了证明，之后再进行开方

这个改成非交互时用的c = Hash(b1…b5, g, h, α, β, δ)

正确与否还需验证

出自 2018 IEEE Symposium on Security and Privacy： Doubly-Efficient zkSNARKs Without Trusted Setup的Figure 5

② 多项式幂证明

(出自12年这个同一个文章)

Common reference string: pk, ck.
Statement: $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_m \in \mathbb{H}^n$, $C \in \mathbb{H}$, and $\boldsymbol{c}_A \in \mathbb{G}^m$
Prover's witness: $A = \{\boldsymbol{a}_j\}_{j=1}^m \in \mathbb{Z}_q^{n \times m}$, $\boldsymbol{r} \in \mathbb{Z}_q^m$, and $\rho \in \mathbb{Z}_q$ such that

$$C = \mathcal{E}_{pk}(1;\rho)\prod_{i=1}^m \boldsymbol{C}_i^{\boldsymbol{a}_i} \qquad \text{and} \qquad \boldsymbol{c}_A = \mathrm{com}_{ck}(A;\boldsymbol{r})$$

Initial message: Pick $\boldsymbol{a}_0 \leftarrow \mathbb{Z}_q^n, r_0 \leftarrow \mathbb{Z}_q$, and $b_0, s_0, \tau_0 \ldots, b_{2m-1}, s_{2m-1}, \tau_{2m-1} \leftarrow \mathbb{Z}_q$ and set $b_m = 0, s_m = 0, \tau_m = \rho$. Compute for $k = 0, \ldots, 2m-1$

$$c_{A_0} = \mathrm{com}_{ck}(\boldsymbol{a}_0; r_0)\,,\ c_{B_k} = \mathrm{com}_{ck}(b_k; s_k)\,,\ \boxed{E_k = \mathcal{E}_{pk}(G^{b_k};\tau_k)\prod_{\substack{i=1,j=0\\ j=(k-m)+i}}^{m,m} \boldsymbol{C}_i^{\boldsymbol{a}_j}}$$

Send: $c_{A_0}, \{c_{B_k}\}_{k=0}^{2m-1}, \{E_k\}_{k=0}^{2m-1}$.
Challenge: $x \leftarrow \mathbb{Z}_q^*$.
Answer: Set $\boldsymbol{x} = (x, x^2, \ldots, x^m)^T$ and compute

$$\boldsymbol{a} = \boldsymbol{a}_0 + A\boldsymbol{x} \qquad r = r_0 + \boldsymbol{r}\cdot\boldsymbol{x} \qquad b = b_0 + \sum_{k=1}^{2m-1} b_k x^k$$

$$s = s_0 + \sum_{k=1}^{2m-1} s_k x^k \qquad \tau = \tau_0 + \sum_{k=1}^{2m-1} \tau_k x^k\,.$$

Send: $\boldsymbol{a}, r, b, s, \tau$.
Verification: Check $c_{A_0}, c_{B_0}, \ldots, c_{B_{2m-1}} \in \mathbb{G}$, and $E_0, \ldots, E_{2m-1} \in \mathbb{H}$, and $\boldsymbol{a} \in \mathbb{Z}_q^n$, and $r, b, s, \tau \in \mathbb{Z}_q$, and accept if $c_{B_m} = \mathrm{com}_{ck}(0;0)$ and $E_m = C$, and

$$c_{A_0}\boldsymbol{c}_A^{\boldsymbol{x}} = \mathrm{com}_{ck}(\boldsymbol{a}; r) \qquad c_{B_0}\prod_{k=1}^{2m-1} c_{B_k}^{x^k} = \mathrm{com}_{ck}(b;s)$$

$$E_0 \prod_{k=1}^{2m-1} E_k^{x^k} = \mathcal{E}_{pk}(G^b;\tau)\prod_{i=1}^m \boldsymbol{C}_i^{x^{m-i}\boldsymbol{a}}\,.$$

$$E_k = \prod_{\substack{1\le i,j\le m\\ j=(k-m)+i}} \boldsymbol{C}_i^{\boldsymbol{a}_j}\,,$$

where $E_m = C$. To visualize this consider the following matrix

(Ek是通过这个矩阵来算的)

$$\begin{array}{cc} & \begin{pmatrix} \boldsymbol{a}_1 & \ldots & \boldsymbol{a}_m \end{pmatrix} \\ \begin{pmatrix} \boldsymbol{C}_1 \\ \boldsymbol{C}_2 \\ \vdots \\ \boldsymbol{C}_m \end{pmatrix} & \begin{pmatrix} \boldsymbol{C}_1^{\boldsymbol{a}_1} & & \ddots & \boldsymbol{C}_1^{\boldsymbol{a}_m} \\ \boldsymbol{C}_2^{\boldsymbol{a}_1} & & \ddots & \boldsymbol{C}_2^{\boldsymbol{a}_m} \\ \ddots & \ddots & \ddots & \ddots \\ \boldsymbol{C}_m^{\boldsymbol{a}_1} & & \ddots & \boldsymbol{C}_m^{\boldsymbol{a}_m} \end{pmatrix} \begin{matrix} E_{2m-1} \\ \vdots \\ E_{m+1} \end{matrix} \\ & E_1 \quad \ldots \quad E_{m-1} \quad E_m \end{array}$$

Challenge x改成非交互式时，通过Hash(g, h, E0…E2m-1)正确与否还需验证

算法实现

算法流程：

Prover输入:
1 公共基点 g, h
2 承诺个数m
3 多个原始承诺 C_array
4 变换序列 pi
5 多个随机数 rou

Prover生成结果(shuffle_proof):
1 多个原始承诺C_array
2 变换后的承诺序列：C_pie
3 对序列pi的承诺：CA_array
4 挑战：x
5 对pi的pi(x)次方的承诺：CB_array
6 挑战：y,z
7 乘积证明(包含KCO和ProductProof)
8 多项式幂方证明: mul

Verifier验证shuffle_proof结果:
1 计算 $\Pi_{i=1}^{N}(yi + x^i - z)$ 并验证乘积证明
2 计算Cx^(x*i)的乘积验证多项式幂方证明

使用约为1018行golang语言代码实现，算法开源在个人github仓库:

https://github.com/YezzizzeY/shuffle

演示说明

使用四个承诺进行输入：

x,y,z,w = 2,3,2,4

rx - rw = 11, 22, 33, 44

Com(x,rx) = {9548...,8543...}

Com(y,ry) = {1835...,3765...}

Com(z,rz) = {1073...,1036...}

Com(w,rw) = {2826...,8841...}

输入重置的序列为：

pi := []int{2,1,3,0}

重置的4个随机数为随机生成的：

497...

447...

468...

729...

可见输出4个置换了随机数和顺序的承诺：

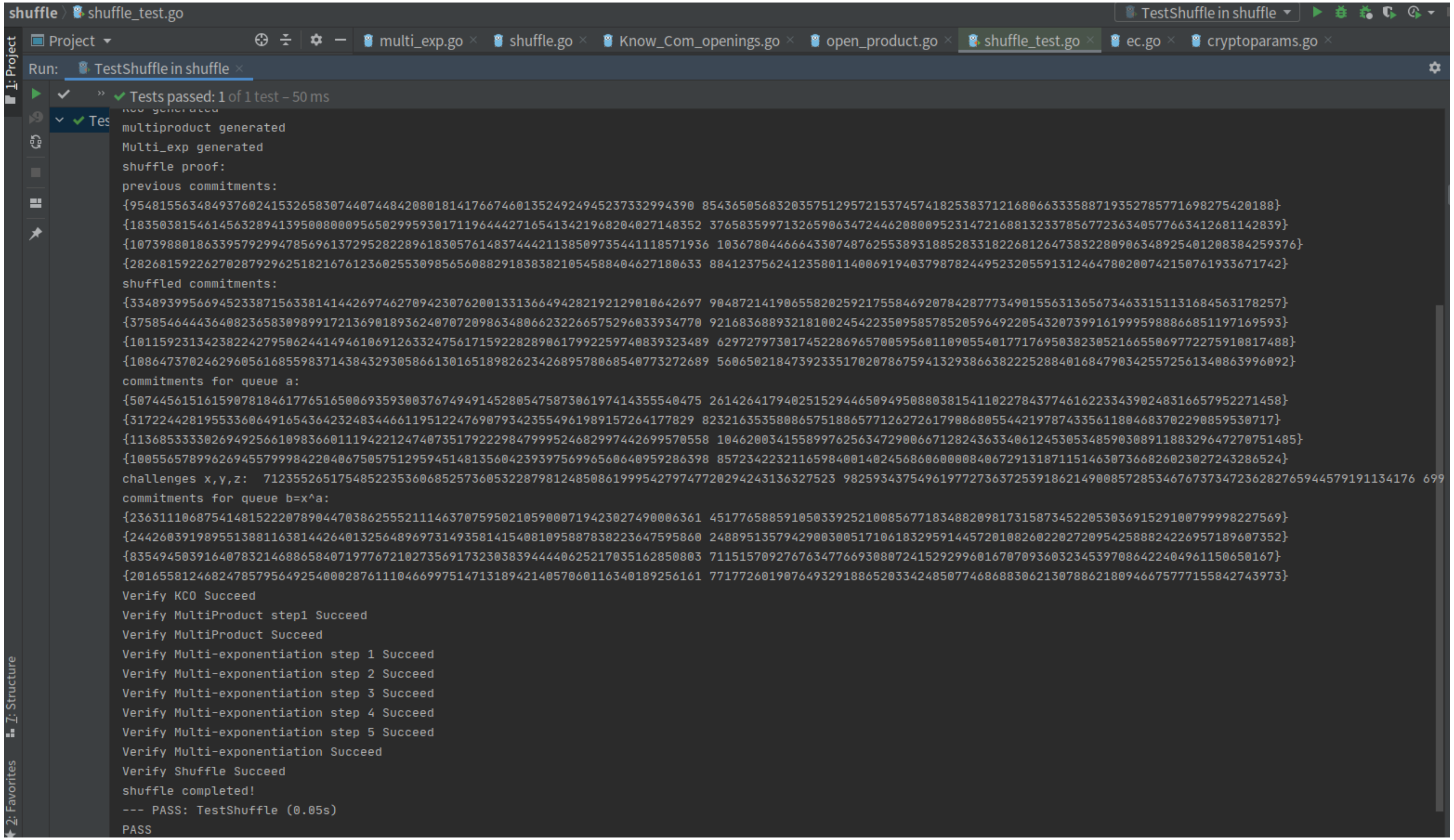
{334...97, 904...57}

{375...70, 921...93}

{101...89, 629...88}

{108...89, 560...92}

,和很多证明并验证通过了



（由于证明和验证数字较多，就不在此放演示截图了。可以下载我的代码进行实验）

其他问题

从交互式到非交互式的正确性不敢保证

算法在工程上的安全性

乘积证明的算法改进

（应该还会有问题。。。）

账户体系下的混淆

通过将一种称为可更新密钥的技术与有效的零知识参数相结合，提出一种新的加密货币Quisquis，它实现了可证明安全的匿名性概念，同时允许用户拒绝参与并存储相对较少的数据量。

出自：EUROCRYPT 2019 Quisquis: A New Design for Anonymous Cryptocurrencies

公私钥：$h_i = g_i^x$

承诺：$com_i = (g_i^r, g^v h_i^r)$ $\longrightarrow$

账户：$acct_i = (pk_i, com_i)$

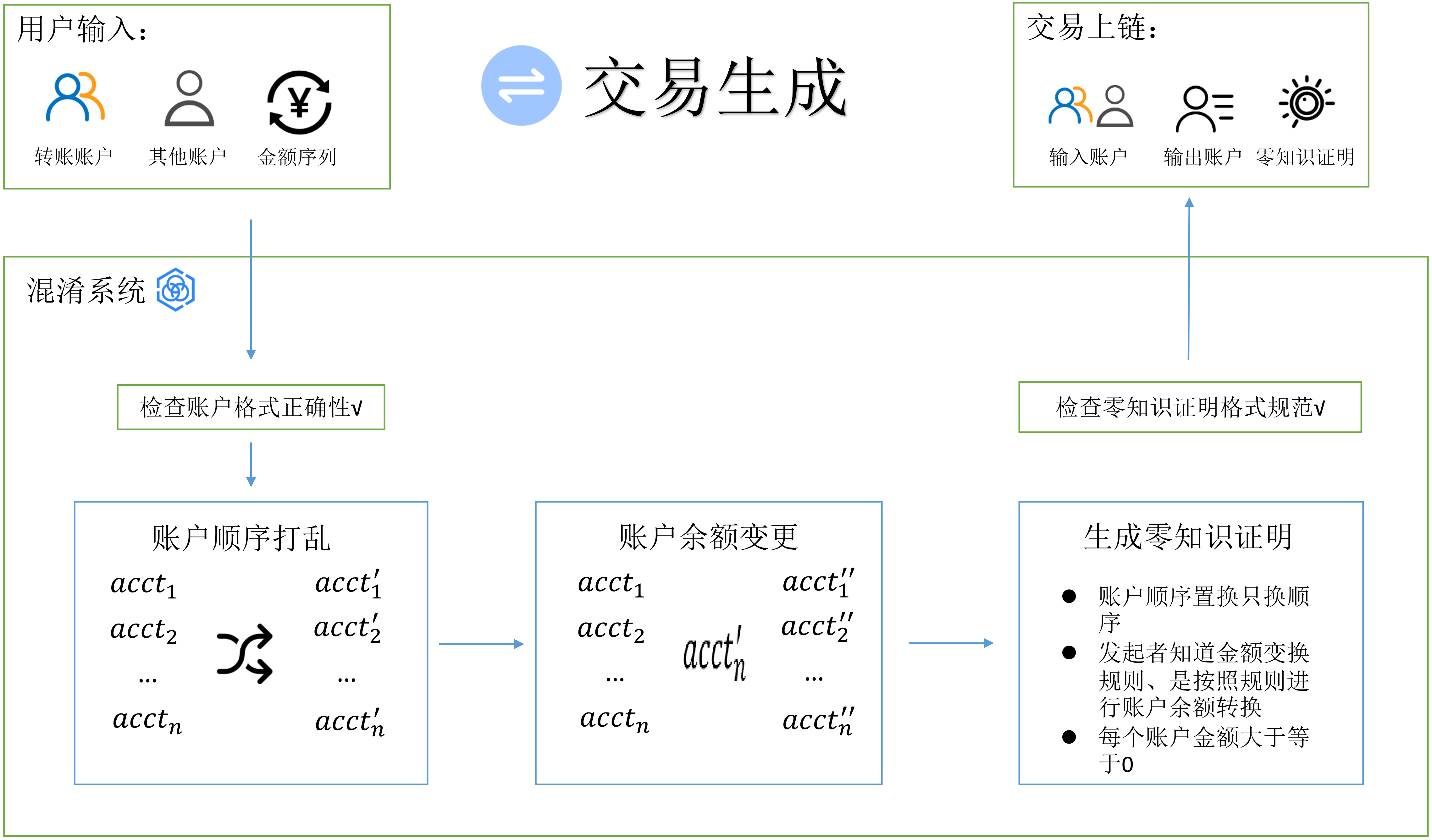
公私钥更新：选取随机数r

$$h_i' = (g_i, h_i)^r$$

更新承诺：要变的金额v1和随机数r1

$$com_i' = (g_i^{r+r1}, g^{v+v1} h_i^{r+r1})$$

账户更新：金额不变(v1=0)、只更新账户数据/金额和账户数据都变/只更新金额


用户输入：
转账账户
其他账户
金额序列
交易生成
交易上链：
输入账户
输出账户
零知识证明
混淆系统
检查账户格式正确性√
检查零知识证明格式规范√
账户顺序打乱
$acct_1$
$acct_2$
…
$acct_n$
$acct'_1$
$acct'_2$
…
$acct'_n$
账户余额变更
$acct_1$
$acct_2$
…
$acct_n$
$acct'_n$
$acct''_1$
$acct''_2$
…
$acct''_n$
生成零知识证明
账户顺序置换只换顺序
发起者知道金额变换规则、是按照规则进行账户余额转换
每个账户金额大于等于0

交易检验

所有人验证：

输入账户未曾被使用

零知识证明正确

检查自己的账户是否被作为输入

否

跳过

是

- 用私钥去寻找自己的账户输出
- 将新的账户状态替换现有的状态

查看自己账户金额是否发生变化

不变：自己的账户仅为匿名集

变化：
用私钥和账户状态遍历出用户自己的新的金额

quisquis零知识证明系统

>交易流程概括：选取一个输入到输出的对应关系ψ（仅发起证明者知道）inputs -> outputs
第一步：ψ1为第一次置换， ψ1(s∗) = 1, ψ1(R∗) = [2, t] ， ψ1(A∗) = [t + 1, n]，其中s*，R*，A*分别为发送者、接收者、匿名集(不参与金额改变，只参与混淆)。根据ψ1，对inputs账户只改变顺序不改变金额，更新变为inputs'。
第二步： 使用所有inputs'的账户公钥生成金额的承诺，并和原账户的承诺乘到一块，改变了账户的金额。inputs'转变为outputs'
第三步： ψ2为第二次置换，只变用户顺序，不变金额。使得 ψ1◦ ψ2= ψ , 将outputs'变为outputs。

>证明系统：第一步和第三步零知识证明方法一样，使用的是**2012年欧密会的文章Ecient Zero-Knowledge Argument for Correctness of a Shuffle**的的证明方法加以稍微改进。对于第二步，结合了四个sigma协议来证明：

1 Σvud：证明对金额的新承诺是根据金额序列v'，使用用户公钥生成，并和原承诺同态加按照实现金额变化
2 Σcom：中途用全局的公钥(g,h)对金额序列v'生成了账户，证明prover知道转账金额序列和随机数
3 Σ_i^{zero}：prover证明自己知道使用每个用户公钥对金额序列v'承诺，并生成acct过程中的随机数
4 ΣNN：中途涉及到的金额都大于0小于V（max）