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 G of prime order q with generator g
- Public key: $pk = y = g^x$
- 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

 C_1

 C_N

 c_N

Input ciphertexts

Permute to get

Re-encrypt them

Output ciphertexts

 c_1, \cdots, c_N

 $c_{\pi(1)}, \cdots, c_{\pi(N)}$

 $C_{i} = c_{\pi(i)} \mathcal{E}_{pk}(1; r_{i})$

 C_1, \cdots, C_N

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:

 $Com(\xrightarrow{n}, r) = g^{v1}g^{v2} \dots g^{r}$

对向量和多个随机数承诺:

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

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: $C, C' \in \mathbb{H}^N$ with N = mn.

Prover's witness: $\pi \in \Sigma_N$ and $\rho \in \mathbb{Z}_q^N$ such that $C' = \mathcal{E}_{pk}(\mathbf{1}; \rho)C_{\pi}$.

Initial message: Pick $r \leftarrow \mathbb{Z}_q^m$, set $a = \{\pi(i)\}_{i=1}^N$ and compute $c_A = \text{com}_{ck}(a; r)$.

Send: c_A

Challenge: $x \leftarrow \mathbb{Z}_q^*$.

Answer Pick $s \in \mathbb{Z}_q^m$, set $b = \{x^{\pi(i)}\}_{i=1}^N$ and compute $c_B = \text{com}_{ck}(b; s)$.

Send: c_B

2

Challenge: $y, z \leftarrow \mathbb{Z}_q^*$.

Answer: Define $c_{-z} = \text{com}_{ck}(-z, \dots, -z; \mathbf{0})$ and $c_D = c_A^y c_B$. Compute d = ya + b, and t = yr + s. Engage in a product argument as described in Sect. 5 of openings $d_1 - z, \dots, d_N - z$ and t such that

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

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

$$oldsymbol{C^x} = \mathcal{E}_{pk}(1;
ho)oldsymbol{C^{\prime b}}$$
 and $oldsymbol{c}_B = \mathrm{com}_{ck}(oldsymbol{b};oldsymbol{s})$

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 c_B .

Verification: The verifier checks $c_A, c_B \in \mathbb{G}^m$ and computes c_{-z}, c_D as described above and computes $\prod_{i=1}^N (yi + x^i - z)$ and C^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 $\rho \in \mathbb{Z}_q^N$ such that $C' = \mathcal{E}_{pk}(\mathbf{1}; \rho)C_{\pi}$.

改成非交互式时,用的

 $x = Hash(g,h, \xrightarrow{CA})$

 $y = Hash(g,h, \xrightarrow{CA}, \xrightarrow{CB})$

 $Z = Hash(g,h, \xrightarrow{CA}, \xrightarrow{CB}, y)$

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 $c_i = \text{com}(\boldsymbol{x}_i; r_i)$. 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** \rightarrow **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** \leftarrow **V:** The verifier sends the prover a random challenge $e \leftarrow \mathbb{Z}_p$.
- **P** \rightarrow **V:** The prover answers the challenge with z, s computed as $z = \sum_{i=0}^{m} e^{i} x_{i}$ and $s = \sum_{i=0}^{m} e^{i} r_{i}$.
- V: The verifier accepts the argument if $\prod_{i=0}^{m} c_i^{e^i} = \text{com}_{ck}(z;s)$.

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 $b_1, \ldots, b_5 \stackrel{\$}{\leftarrow} \{1, \ldots, q_G\}$ 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 $c \stackrel{\$}{\leftarrow} \{1, \dots, q_G\}$
- 3. \mathcal{P} sends

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

4. V checks that

$$\alpha \odot X^c \stackrel{?}{=} g^{z_1} \odot h^{z_2}$$
 (7)

$$\beta \odot Y^c \stackrel{?}{=} g^{z_3} \odot h^{z_4}$$
 (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: $C_1, \ldots, C_m \in \mathbb{H}^n$, $C \in \mathbb{H}$, and $c_A \in \mathbb{G}^m$

Prover's witness: $A=\{a_j\}_{j=1}^m\in\mathbb{Z}_q^{n\times m}, r\in\mathbb{Z}_q^m, \text{ and } \rho\in\mathbb{Z}_q \text{ such that }$

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

Initial message: Pick $a_0 \leftarrow \mathbb{Z}_q^n$, $r_0 \leftarrow \mathbb{Z}_q$, and $b_0, s_0, \tau_0 \dots, b_{2m-1}, s_{2m-1}, \tau_{2m-1} \leftarrow \mathbb{Z}_q$ and set $b_m = 0, s_m = 0, \tau_m = \rho$. Compute for $k = 0, \dots, 2m-1$

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

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 $\mathbf{x} = (x, x^2, \dots, x^m)^T$ and compute

$$a = a_0 + Ax$$
 $r = r_0 + r \cdot x$ $b = b_0 + \sum_{k=1}^{2m-1} b_k x^k$ $s = s_0 + \sum_{k=1}^{2m-1} s_k x^k$ $\tau = \tau_0 + \sum_{k=1}^{2m-1} \tau_k x^k$.

Send: a, r, b, s, τ .

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 $a \in \mathbb{Z}_q^n$, and $r, b, s, \tau \in \mathbb{Z}_q$, and accept if $c_{B_m} = \operatorname{com}_{ck}(0; 0)$ and $E_m = C$, and

$$c_{A_0} \mathbf{c}_A^{\mathbf{x}} = \text{com}_{ck}(\mathbf{a}; r) \qquad c_{B_0} \prod_{k=1}^{2m-1} c_{B_k}^{x^k} = \text{com}_{ck}(b; s)$$
$$E_0 \prod_{k=1}^{2m-1} E_k^{x^k} = \mathcal{E}_{pk}(G^b; \tau) \prod_{i=1}^m \mathbf{C}_i^{x^{m-i} \mathbf{a}}.$$

$$E_k = \prod_{\substack{1 \le i, j \le m \\ j = (k-m)+i}} C_i^{a_j} ,$$

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

$$(\mathbf{E}$$
k是通过这个 $\begin{pmatrix} C_1 \ C_2 \ dots \ C_m \end{pmatrix} \begin{pmatrix} C_1^{a_1} & \ddots & C_1^{a_m} \ C_2^{a_1} & \ddots & C_2^{a_m} \ \ddots & \ddots & \ddots & \ddots \ C_m^{a_1} & \ddots & C_m^{a_m} \end{pmatrix} E_{2m-1} \ dots \ E_1 \ \dots \ E_{m-1} \ E_m$

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

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

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 $\prod_{i=1}^{N} (yi + x^i z)$ 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
```

演示说明

```
🍍 TestShuffle in shuffle 🔻 🕨 🇯 ち 🗣 🔻
shuffle 🕽 👺 shuffle_test.go
                                 😌 🛨 🔯 🛑 🗑 multi_exp.go × 🔞 shuffle.go × 🐧 Know_Com_openings.go × 🐧 open_product.go × 🐧 shuffle_test.go × 🐧 ec.go ×
              Tests passed: 1 of 1 test – 50 ms
              multiproduct generated
               Multi_exp generated
               shuffle proof:
              previous commitments:
 ==
               \{9548155634849376024153265830744074484208018141766746013524924945237332994390\ 85436505683203575129572153745741825383712168066333588719352785771698275420188\}
               \{18350381546145632894139500800095650299593017119644427165413421968204027148352\ 37658359971326590634724462080095231472168813233785677236340577663412681142839\}
               \{107398801863395792994785696137295282289618305761483744421138509735441118571936\ 103678044666433074876255389318852833182268126473832280906348925401208384259376\}
               $1.9268159296270287929625182167612360255309856560882918383821054588404627180633
$1.8268159296270287929625182167612360255309856560882918383821054588404627180633
$1.8268159296270287929625182167612360255309856560882918383821054588404627180633
$1.826815929627028792962518216761236025591826679180633
               shuffled commitments
               \{33489399566945233871563381414426974627094230762001331366494282192129010642697 90487214190655820259217558469207842877734901556313656734633151131684563178257\}
               \{37585464443640823658309899172136901893624070720986348066232266575296033934770
92168368893218100245422350958578520596492205432073991619995988866851197169593\}
               \{101159231342382242795062441494610691263324756171592282890617992259740839323489\ 62972797301745228696570059560110905540177176950382305216655069772275910817488\}
               \{108647370246296056168559837143843293058661301651898262342689578068540773272689\ 56065021847392335170207867594132938663822252884016847903425572561340863996092\}
               commitments for queue a:
               \{50744561516159078184617765165006935930037674949145280547587306197414355540475\ 26142641794025152944650949508803815411022784377461622334390248316657952271458\}
               \{3172244281955336064916543642324834466119512247690793423554961989157264177829,82321635358086575188657712627261790868055442197874335611804683702290859530717\}
               \{113685333302694925661098366011194221247407351792229847999524682997442699570558\ 104620034155899762563472900667128243633406124530534859030891188329647270751485\}
               \{100556578996269455799984220406750575129594514813560423939756996560640959286398\ 8572342232116598400140245686060000840672913187115146307366826023027243286524\}
               challenges x.v.z: 71235526517548522353606852573605322879812485086199954279747720294243136327523 98259343754961977273637253918621490085728534676737347236282765944579191134176 694
               commitments for queue b=x^a
               \{236311110687541481522207890447038625552111463707595021059000719423027490006361\ 45177658859105033925210085677183488209817315873452205303691529100799998227569\}
               \{24426039198955138811638144264013256489697314935814154081095887838223647595860,24889513579429003005171061832959144572010826022027209542588824226957189607352\}
               \{83549450391640783214688658407197767210273569173230383944446625217035162850803,7115157092767634776693080724152929960167070936032345397086422404961150650167\}
               \{20165581246824785795649254000287611104669975147131894214057060116340189256161\ 77177260190764932918865203342485077468688306213078862180946675777155842743973\}
               Verify KCO Succeed
               Verify MultiProduct step1 Succeed
               Verify MultiProduct Succeed
               Verify Multi-exponentiation step 1 Succeed
               Verify Multi-exponentiation step 2 Succeed
               Verify Multi-exponentiation step 3 Succeed
               Verify Multi-exponentiation step 4 Succeed
               Verify Multi-exponentiation step 5 Succeed
               Verify Multi-exponentiation Succeed
               shuffle completed!
               --- PASS: TestShuffle (0.05s)
```

(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: $h_i = g_i^x$

账户: $acct_i = (pk_i, com_i)$

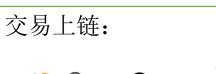
公私钥更新: 选取随机数r $h'_i = (g_i, h_i)^r$

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













输入账户 输出账户 零知识证明

检查账户格式正确性V

检查零知识证明格式规范V

账户顺序打乱

 $acct'_1$ $acct_1$ $acct'_2$ $acct_2$

 $acct_n$

 $acct'_n$

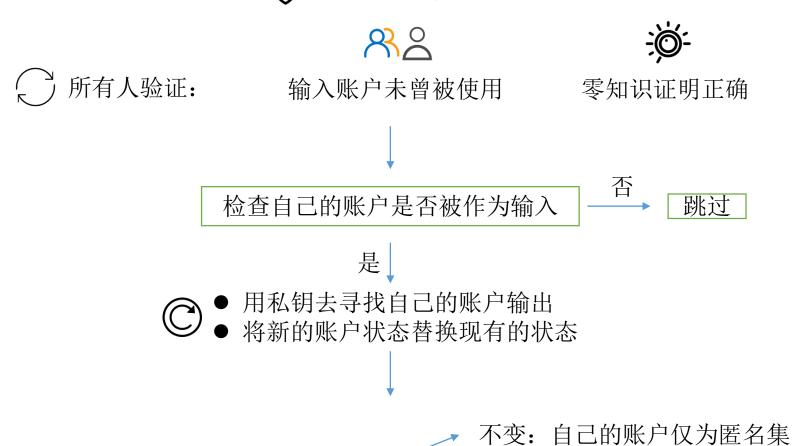
账户余额变更

 $acct_1''$ $acct_1$ $acct_2''$ $acct_2$ $acct''_n$ $acct_n$

生成零知识证明

- 账户顺序置换只换顺
- 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

心交易检验



Q 查看自己账户金额是否发生变化

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: ψ 1为第一次置换, ψ 1(s*) = 1, ψ 1(R*) = [2, t] , ψ 1(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 ψ 1,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: ψ 2为第二次置换,只变用户顺序,不变金额。使得 ψ 1° ψ 2= 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

 $3\Sigma_{i}^{zero}$: 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

4ΣNN: 中途涉及到的金额都大于0小于V(max)