

CÔNG NGHỆ BLOCKCHAIN VÀ ỨNG DỤNG

BÁO CÁO ĐỒ ÁN: THRESHOLD MULTI SIGNATURE – 18TN

I. Thành viên:

18120019 – Nguyễn Hoàng Dũng

18120130 – Võ Anh Khoa

II. Nội dung đồ án:

Xây dựng thuật toán đa chữ ký số để áp dụng trên các hệ thống phân tán như blockchain. Đồng thời cho phép đối với nhóm có N user, chỉ cần tối thiểu K signer bất kì là có thể tạo ra được cùng 1 chữ ký số dưới định danh của cả nhóm.

Các phép toán trên chữ ký số được thực hiện trên tập hợp các điểm thuộc Elliptic Curve $E(\mathbb{Z}_p)$ có bậc của nhóm là n và phần tử sinh G, thỏa phương trình như sau:

$$y^2 = x^3 + ax + b \mod p$$

III. Thuật toán đa chữ ký MuSig đối với nhóm gồm N user:

Tham khảo từ [whitepaper](#) của TariLabs University.

IV. Cải tiến thuật toán MuSig với yêu cầu tối thiểu K signer:

Round 1:

Mỗi user thứ i khởi tạo ngẫu nhiên một đa thức bí mật có bậc k – 1:

$$f_i(x) = a_{i0} + a_{i1}x + \dots + a_{ik-1}x^{k-1} \mod n$$

Trong đó hệ số a_{i0} chính là private key của user thứ i. Public key của user thứ i sẽ là:

$$P_i = G \cdot a_{i0}$$

Như vậy, đa thức bí mật của cả nhóm sẽ là tổng các đa thức bí mật của mỗi user:

$$f(x) = \sum_{i=1}^n f_i(x)$$

Trong đó hệ số $a_0 = f(0)$ của đa thức $f(x)$ chính là private key của cả nhóm. Không user nào có được bất kì thông tin gì về các hệ số của đa thức bí mật $f(x)$

Round 2:

Gọi L là hash của tất các public key của thành viên trong nhóm:

$$L = H(P_1, P_2, \dots, P_n)$$

Từ public key của chính mình, mỗi user cần tính challenge c_i của bản thân:

$$c_i = H_{agg}(L, P_i)$$

Khi đó mỗi user sẽ sở hữu một challenge private key như sau:

$$x_i = a_{i0} \cdot c_i$$

Public key tổng hợp của nhóm sẽ là:

$$P = \sum_{i=1}^n (P_i \cdot c_i)$$

Round 3:

Mỗi user thứ i sẽ broadcast phiên bản obfuscated của đa thức bí mật $f_i(x)$ bằng cách nhân từng hệ số với G :

$$(G \cdot a_{i0}), (G \cdot a_{i1}), \dots, (G \cdot a_{ik-1})$$

Mỗi user thứ i sẽ thông qua kênh truyền bí mật để gửi bộ nghiệm bí mật $(j, f'_i(j))$ đến user thứ j . Trong đó:

$$f'_i(x) = x_i + a_{i1}x + \dots + a_{ik-1}x^{k-1} \mod n$$

User j có thể kiểm tra tính chính xác của bộ nghiệm nhận được từ user i bằng cách sử dụng các hệ số đã được obfuscated của đa thức $f_i(x)$:

$$f_i(x) \cdot G = a_{i0} \cdot G + a_{i1}x \cdot G + \dots + a_{ik-1}x^{k-1} \cdot G \mod n$$

Trong đó, obfuscated của hệ số tự do chính là public key của user i và ta cần nhân thêm với giá trị challenge c_i của user i :

$$f'_i(x) \cdot G = c_i \cdot a_{i0} \cdot G + a_{i1}x \cdot G + \dots + a_{ik-1}x^{k-1} \cdot G \mod n$$

$$f'_i(x) \cdot G = c_i \cdot P_i + a_{i1}x \cdot G + \dots + a_{ik-1}x^{k-1} \cdot G \mod n$$

Sau khi quá trình trao đổi thông tin bí mật giữa các user hoàn tất, user thứ j sẽ nắm giữ bộ nghiệm thứ j của đa thức bí mật $f'(x)$ của cả nhóm:

$$y'_j = f'(j) = \sum_{i=1}^n f'_i(j)$$

Mặt khác, $f'(0)$ chính là tổng của các challenge private key:

$$f'(0) = \sum_{i=1}^n x_i = \sum_{i=1}^n (a_{i0} \cdot c_i)$$

Đa thức $f'(x)$ là một đa thức bậc $k - 1$ gồm k hệ số. Mỗi user trong nhóm N user đã giữ 1 bộ nghiệm bí mật (j, y'_j) . Như vậy, bằng cách sử dụng công thức nội suy Lagrange, chỉ cần k user bất kì kết hợp k bộ nghiệm với nhau là có thể xây dựng lại được đa thức $f'(x)$ ban đầu và tính được $f'(0)$

$$f'(0) = \sum_{i=1}^k \left[y'_i \prod_{j \neq i} (-j)(i - j)^{-1} \right]$$

Mỗi user thứ i trong nhóm N người sẽ có thể khôi phục được 1 phần trong k phần của group private key ứng với group public key tổng hợp P :

$$\pi_i = y'_i \prod_{j \neq i} (-j)(i - j)^{-1}$$

Round 4:

K user bất kì trong nhóm N người ban đầu muốn cùng kí một văn bản m . K người này có thể xây dựng lại được public key của nhóm như sau:

$$P = \sum_{i=1}^k (G \cdot \pi_i)$$

K user sẽ ngẫu nhiên K số nonce r_i và tính $R_i = G \cdot r_i$ và broadcast $t_i = H(R_i)$. Sau khi nhận đủ $(k - 1)$ t_i , mỗi user sẽ đồng loạt public R_i và kiểm tra tất cả các $H(R_i)$ có ứng với các t_i đã nhận được trước đó. Nếu thỏa, ta sẽ có bộ chữ kí (R, s) :

$$R = \sum_{i=1}^k R_i$$

$$e = H_{sig}(P, R, m)$$

$$s = \sum_{i=1}^k (r_i + e \cdot \pi_i)$$

Verify Signature:

$$G \cdot s = R + P \cdot e$$

$$G \cdot \sum_{i=1}^k (r_i + e \cdot \pi_i) = \sum_{i=1}^k R_i + e \cdot \sum_{i=1}^n (P_i \cdot c_i)$$

$$G \cdot \sum_{i=1}^k r_i + e \cdot G \cdot \sum_{i=1}^k \pi_i = G \cdot \sum_{i=1}^k r_i + e \cdot G \cdot \sum_{i=1}^n (a_{i0} \cdot c_i)$$

$$\sum_{i=1}^k r_i + e \cdot \sum_{i=1}^k \pi_i = \sum_{i=1}^k r_i + e \cdot \sum_{i=1}^n (a_{i0} \cdot c_i)$$

$$\sum_{i=1}^k r_i + e \cdot f'(0) = \sum_{i=1}^k r_i + e \cdot f'(0)$$

V. Cài đặt:

Nhóm tiến hành cài đặt và demo để kiểm tra tính đúng đắn của thuật toán, sử dụng ngôn ngữ Python và thư viện django hỗ trợ tương tác. Có 2 class chính cần cài đặt là class **Member** định nghĩa các đối tượng trong tổ chức và class **Party** định nghĩa tổ chức và thao tác kí giữa các Member.

```
class Member():
    def __init__(self, _name: str, _min: int, _max: int):
        self.name = _name
        self.p_once = E.INF
        self.s_once = 0
        self.p_key = E.INF
        self.s_poly = [0] * _min
        self.x_key = 0
        self.share = [0] * _max
        self.chall = 0
        self.secret = 0
        self.online = 0
```

Trong đó, mỗi member sẽ có:

- `_name`: id định danh các user trong tổ chức
- `_min`: số lượng hệ số của đa thức bí mật
- `_max`: số lượng share secret tối đa nhận được (tương ứng số member trong party)

- p_once: public nonce
- s_once: secret nonce
- p_key: public key
- s_poly: đa thức bí mật có [0] là private key
- x_key: challenge private key
- share: mảng chứa các nghiệm bí mật tính được từ đa thức bí mật của các user còn lại
- chall: giá trị challenge được tính từ $H(L, P_i)$
- online: biến xác định xem member đó có tham gia vào quá trình sign hay không

```
class Party():
    def __init__(self, _min: int, _max: int):
        self.people = [fool]
        self.signable = False
        self.now_size = 1
        self.max_size = 1 + _max
        self.min_size = 0 + _min
        self.storage = []
```

Trong đó:

- _min: số lượng người cần tối thiểu cho quá trình kí
- _max: số lượng người tối đa trong party
- signable: biến xác định xem party có đủ điều kiện để tạo chữ kí hay không (demo)
- people: mảng chứa các member (có fool là biến dummy, để set index bắt đầu là 1)
- storage: mảng lưu các thôn tin được broadcast giữa các member trong tổ chức

Các hàm quan trọng:

```
def create_user(self, _name: str):
    if self.now_size == self.max_size: return False
    user = Member(_name, self.min_size, self.max_size)
    self.people.append(user)
    self.now_size = len(self.people)
    return self.people[self.now_size - 1].gen_private_poly()
```

Khi 1 member được thêm vào party, member đó sẽ tự sinh ra đa thức bí mật có private key tại [0] và suy ra public key. Việc return chỉ mang tính chất demo và hiển thị lên giao diện để phục vụ mục đích kiểm tra thuật toán.

```
def setup_group(self):
    if self.now_size != self.max_size:
        return False
    self.set_private_chall()
    self.broadcast_obfuscated_poly()
    self.each_share_secret()
    self.signable = self.check_share_secret()
    return self.signable
```

Khi đã đủ số lượng member. Đầu tiên mỗi người sẽ tự tính challenge của bản thân. Giá trị challenge sẽ được nhân với private key để tạo thành challenge private key (master key)

```
def set_private_chall(self):
    group_addr = self.get_all_address()
    group_hash = hash_multiset(group_addr)
    for i in range(1, self.max_size):
        self.people[i].set_master_key(group_hash)
```

```
def set_master_key(self, group_hash: bytes):
    _chall = hash_aggregate(group_hash, self.p_key)
    _chall = bytes_to_long(_chall)
    self.chall = _chall
    self.x_key = self.s_poly[0] * _chall % N
```

Sau đó là broadcast các hệ số của đa thức bí mật của bản thân (đã obfuscated)

```
def broadcast_obfuscated_poly(self):
    self.storage = [[]]
    for i, user in enumerate(self.people):
        if i == 0: continue
        obs_poly = user.obfuscate_poly()
        self.storage.append(obs_poly)
```

```
def obfuscate_poly(self) -> list[Point]:
    result = []
    for coef in self.s_poly:
        obfs = coef * G
        result.append(obfs)
    result[0] *= self.chall
    return result
```

Tiếp đến là tính các nghiệm bí mật và trao đổi trực tiếp qua kênh truyền bí mật. Việc trao đổi thông tin giữa 2 user chỉ được code minh họa ở mức demo, đủ để kiểm tra tính đúng đắn của thuật toán. Trên mạng blockchain thực tế, sẽ là kết nối peer to peer giữa 2 node.

```
def each_share_secret(self):
    for j in range(1, self.max_size):
        for i in range(1, self.max_size):
            y_value = self.people[i].gen_secret_share(j)
            self.people[j].recv_secret_share(y_value, i)
```

```
def gen_secret_share(self, j: int) -> int:
    order = len(self.s_poly)
    y_value = self.x_key
    for i in range(1, order):
        y_value += self.s_poly[i] * j % N
        j = j * j
    return y_value % N

def recv_secret_share(self, y_value: int, i: int):
    self.share[i] = y_value
```

Cuối cùng, các member sẽ kiểm tra tính chính xác của các nghiệm bí mật đã nhận được bằng cách sử dụng các thông tin obfuscated về đa thức đã được broadcast

```
def check_share_secret(self):
    for i, user in enumerate(self.people):
        if i == 0: continue
        if True != user.verify_obfuscation(i, self.storage):
            return False
        user.sum_secret_share()
    return True
```

```
def verify_obfuscation(self, index: int, obs: list[list[Point]]):
    size = len(self.share)
    for i in range(1, size):
        if i == index: continue
        y = obs[i][0]
        x = index
        for j in range(1, len(obs[i])):
            y = y + x * obs[i][j]
            x = x * x
        if y != self.share[i] * G: return False
    return True
```

Sau khi có đủ các nghiệm bí mật, và xác nhận là đúng, mỗi user sẽ tính tổng để tìm ra nghiệm bí mật của đa thức tổng hợp

```
def sum_secret_share(self):
    size = len(self.share)
    self.secret = 0
    for i in range(1, size):
        self.secret += self.share[i]
        self.secret %= N
```

Từ 1 nghiệm bí mật, mỗi user sẽ tính được 1 phần trong k phần của private key tương ứng với public key nhóm, bằng cách sử dụng công thức nội suy Lagrange

```
def lagrange_interpolate(self, index: int, group: list[int]):
    f0 = 1
    for other in group:
        if other == index: continue
        iv = pow(index - other, -1, N)
        f0 = f0 * iv * - other % N
    f0 = f0 * self.secret % N
    return f0
```

Quá trình kí sẽ diễn ra như sau:

```
def sign_message(self, message: str):
    subset = self.count_signers()
    if len(subset) < self.min_size: return False
    subset = subset[:self.min_size]

    p = self.rebuild_group_pkey(subset)
    assert p == self.true_group_pkey()

    r = self.public_group_nonce(subset)
    e = hash_signature(p, r, message)
    s = self.create_signature(e, subset)

    assert True == self.clear_all_nonce()
    return (p, r, s)
```


Đầu tiên là kiểm tra số lượng member đủ điều kiện để kí. User đó phải online và có public nonce khác rỗng. Nếu đủ số lượng người như yêu cầu tối thiểu thì tiếp tục.

```
def count_signers(self) -> list[int]:
    if self.signable == False: return []
    result = []
    for i, user in enumerate(self.people):
        if user.p_once != E.INF and user.online == True:
            result.append(i)
    return result
```

Tạo lại từng phần của group public key chỉ dựa vào k người.

```
def rebuild_group_pkey(self, subset: list[int]) -> Point:
    pk_group = E.INF
    for signer in subset:
        pi_piece = self.people[signer].get_pkey_piece(signer, subset)
        pk_group = pk_group + pi_piece
    return pk_group
```

```
def get_pkey_piece(self, index: int, group: list[int]) -> Point:
    f0 = self.lagrange_interpolate(index, group)
    return G * f0
```

Để kiểm tra tính đúng đắn của thuật toán, ta sẽ so sánh lại với giá trị public key được tạo khi đủ N người. Trên thực tế, từ các thông tin đã được broadcast, mỗi user hoàn toàn có thể tự tính được public key để phục vụ cho quá trình tạo chữ kí.

```
def true_group_pkey(self) -> Point:
    pk_group = E.INF
    for user in self.people:
        pk_group = pk_group + user.p_key * user.chall
    return pk_group
```

Sau đó là tính nonce R tổng hợp từ k người tham gia vào quá trình kí

```
def public_group_nonce(self, subset: list[int]) -> Point:
    r_nonce = E.INF
    for signer in subset:
        r_nonce += self.people[signer].p_once
    return r_nonce
```

Mỗi user sẽ tính 1 phần của chữ kí s và kết hợp để tạo ra chữ kí duy nhất dưới định danh của toàn bộ nhóm. Việc kí sẽ yêu cầu thêm common challenge e

```
def create_signature(self, e: bytes, subset: list[int]):  
    v_sign = 0  
    for signer in subset:  
        v_sign += self.people[signer].sign_challenge(e, signer, subset)  
        v_sign %= N  
    return v_sign
```

```
def sign_challenge(self, e: bytes, index: int, group: list[int]):  
    f0 = self.lagrange_interpolate(index, group)  
    Si = self.s_once + f0 * bytes_to_long(e) % N  
    return Si % N
```

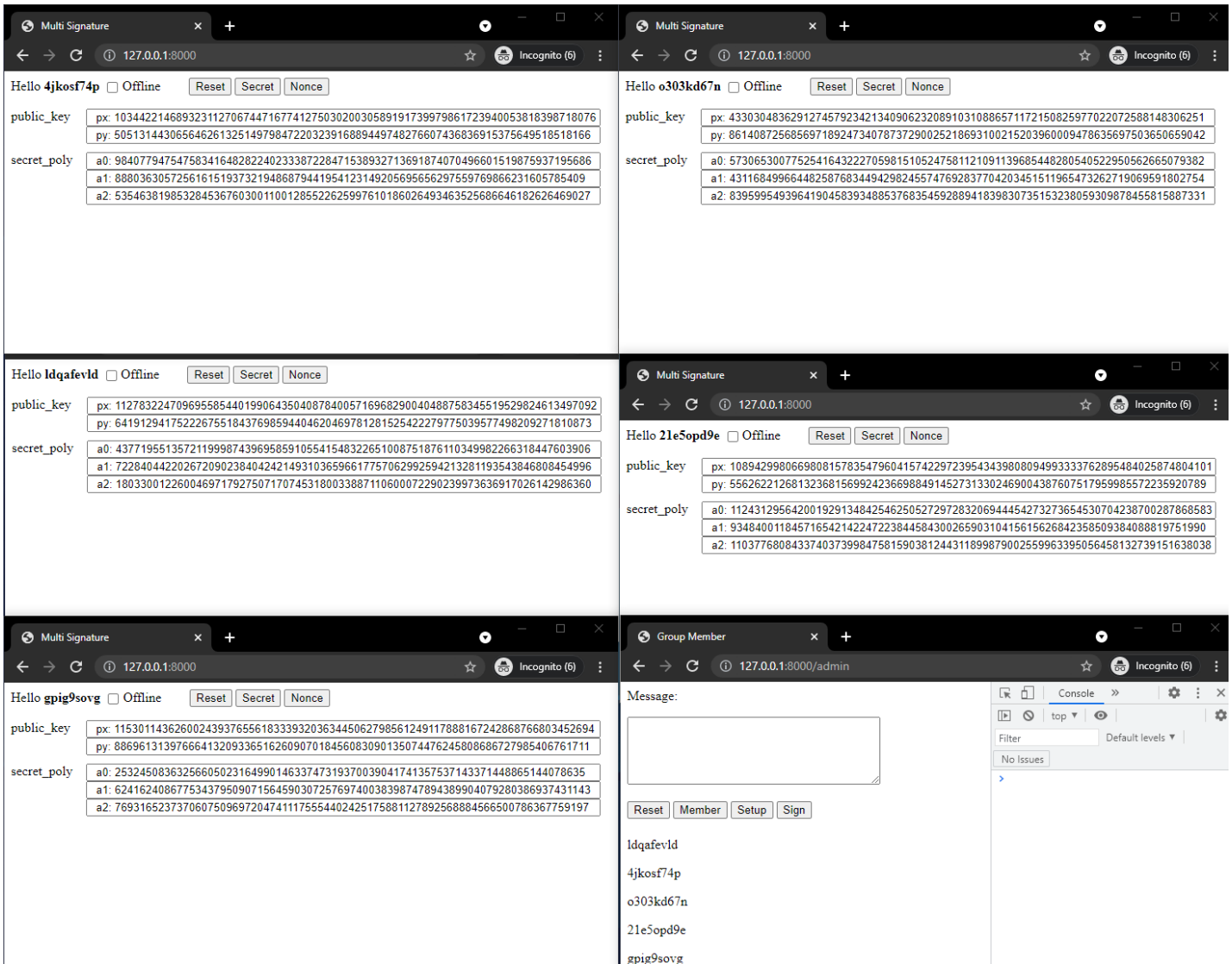
VI. DEMO:

Nhóm sẽ demo trên đường cong NIST P256

Đầu tiên chạy lệnh: python main.py runserver

```
PS D:\Aspire\Desktop\Github\multisig> python .\main.py runserver  
Watching for file changes with StatReloader  
Performing system checks...  
  
System check identified no issues (0 silenced).  
July 10, 2021 - 18:03:43  
Django version 3.2.5, using settings 'mysite.settings'  
Starting development server at http://127.0.0.1:8000/  
Quit the server with CTRL-BREAK.
```

Sau đó spawn ra 5 cửa sổ tương ứng với 5 member:



Trang user sẽ có các tương tác như:

- Reset: Nhấn để tạo mới user trong nhóm, xóa user cũ khỏi nhóm
- Secret: Nhấn để xem các giá trị secret như private key và nghiệm bí mật
- Nonce: Nhấn để tạo mới nonce sau mỗi lần kí

Trang admin được làm ra với mục đích demo:

- Member: Nhấn để hiện thông tin về các member
- Reset: Nhấn để xóa toàn bộ member trong nhóm
- Setup: Nhấn để thực hiện quá trình trao đổi và xác minh nghiệm bí mật giữa các user
- Sign: Nhấn để tạo chữ kí cho văn bản đã điền ở phần message box

Multi Signature

127.0.0.1:8000

Incognito (6)

Hello 4jkskf74p Online

Reset Secret Nonce

public_key

px: 103442214689323112706744716774127503020030589191739979861723940053818398718076
py: 50513144306564626132514979847220323916889449748276607436836915375649518518166

secret_poly

a0: 9840779457458341648282240233387228471538932713691874070496601519875937195686
a1: 8880363057256161519373219486879441954123149205695656297559769866231605785409
a2: 53546381985328453676030011001285522625997610186026493463525686646182626469027

secret_pair

sk: 2716946245191302017521144985784792475377342715268599408820046689865986550116
fx: 40927349670949226558667072074385129337880958532384196641922789905916552039589

nonce_pair

s1: 317085059197838427707096748049875237459133454744544849959388848677868191515
rx: 24700296951097816640327627288858375803551574703219162441407159583081134883546
ry: 11145791514340535890122747933036473493771976915224176569485247241434853646054

Multi Signature

127.0.0.1:8000

Incognito (6)

Hello o303kd67n Online

Reset Secret Nonce

public_key

px: 43303048362912745792342134090623208910310886571172150825977022072588148306251
py: 86140872568569718924734078737290025218693100215203960009478635697503650659042

secret_poly

a0: 57306530077525416432227059815105247581121091139685448280540522950562665079382
a1: 43116849966448258768344942982455747692837704203451511965473262719069591802754
a2: 83959954939641904583934885376835459288941839830735153238059309878455815887331

secret_pair

sk: 11407201308834858395663437195855139181176010442752480948231871663456867802862
fx: 6688668923146555874979892489786656468359975651188519056819491037326134211373

nonce_pair

s1: 32890529483610527619305950438590834402663779246603659254258185717597092136653
rx: 2287118327035480959484993661295101022940368255203416092705266872988026186666
ry: 11046507832250632885638286404103782243166646001941322764711041212182977625221

Multi Signature

127.0.0.1:8000

Incognito (6)

Hello ldqafevd Online

Reset Secret Nonce

public_key

px: 112783224709695585440199064350408784005716968290040488758345519529824613497092
py: 6419129417522267551843769859440462046978128152542279775039577498209271810873

secret_poly

a0: 43771955135721199987439695859105541548322651008751876110349882266318447603906
a1: 72284044220267209023840424214931036596617757062992594213281193543846808454996
a2: 18033001226004697179275071707453180033887110600072290239973636917026142986360

secret_pair

sk: 27142360844559567470610737015931586471974852037384548535737619282802149715402
fx: 5912817584405737663469731728720174138483660209808897983491850558476107081497

nonce_pair

s1: 596364042743107088288458237508008829892617777185404288142393943752210662977
rx: 12308887582477764914550277763028657980662649286120697270540507031562722753832
ry: 50454292455738794237007834136753062250483454257251034679063606326682199820034

Multi Signature

127.0.0.1:8000

Incognito (6)

Hello gpig9sovg Offline

Reset Secret Nonce

public_key

px: 115301143626002439376556183339320363445062798561249117888167242868766803452694
py: 88696131397666413209336516260907018456083090135074476245808686727985406761711

secret_poly

a0: 2532450836325660502316499014633747319370039041741357357143371448865144078635
a1: 62416240867753437950907156459030725769740038398747894389904079280386937431143
a2: 76931652373706075096972047411175554402425175881127892568884566500786367759197

secret_pair

sk: 10467495559404117880423046074943291462788991446531219789604510309069218500268
fx: 91639790774416753019866167978278875814001851437484235156010178652052710196017

Group Member

127.0.0.1:8000/admin

Incognito (6)

Message:

Reset Member Setup Sign

ldqafevd
4jkskf74p
o303kd67n
21e5opd9e
gpig9sovg

Filter

Default levels

No Issues

{stat: true} admin:82

Sau khi đã tạo đủ 5 user, nhấn setup ở phần trang admin để quá trình trao đổi nghiệm được diễn ra. Kết quả là {stat: true} báo hiệu đã setup thành công. Khi đó nhấn secret để xem giá trị của 2 secret key quan trọng. Đó là challenge private key (sk) và nghiệm bí mật (fx) tương ứng với x là thứ tự của user được hiện lên trong danh sách.

Quá trình kí sẽ yêu cầu 3 user online và 3 user đó cần tạo nonce. Chọn 3 user bất kì. Sau đó nhập văn bản cần tạo chữ kí.

Message:

multi signature 18TN

Reset

Member

Setup

Sign

ldqafevld	p_key	px: 2791919217030134880261367517096688539463165688461878137297495257516293508965
4jksf74p		py: 10752653959125568890966566729297343219772250331186225368449729616195356057481
o303kd67n	r_once	rx: 25099358978983554848238841593657940143726438794761932320656938045906904881455
		ry: 19162187116871110263897410183426934327846724283887871801900349095524318596225
21e5opd9e	v_sign	s: 28796573032610260297090403943384922118949029022494040632249919345822091842259
gpig9sovg		

Các kết quả trên có thể được xác minh độc lập bằng đoạn code ngắn như sau:

```
2  from demo.curve import *
3
4
5
6  px = 2791919217030134880261367517096688539463165688461878137297495257516293508965
7  py = 10752653959125568890966566729297343219772250331186225368449729616195356057481
8
9  rx = 25099358978983554848238841593657940143726438794761932320656938045906904881455
10 ry = 19162187116871110263897410183426934327846724283887871801900349095524318596225
11
12 s = 28796573032610260297090403943384922118949029022494040632249919345822091842259
13
14
15 P = Point(px, py, E)
16 R = Point(rx, ry, E)
17 e = hash_signature(P, R, 'multi signature 18TN')
18 e = bytes_to_long(e)
19
20
21 print(s * G == R + P * e)
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS D:\Aspire\Desktop\Github\multisig> python .\verify.py
True
PS D:\Aspire\Desktop\Github\multisig> 
```

VII. Tài liệu tham khảo:

[The MuSig Schnorr Signature Scheme](#)

[Threshold Signatures and Accountability](#)

[Shared Secrets and Threshold Signatures](#)