Cryptographic Primitives in DEXON

Po-Chun Kuo and Hao Chung
DEXON Research
2019.1

DEXON Reading Group

1. Overview of DEXON
2. **Cryptographic Primitive in DEXON**
3. DEXON Byzantine Agreement
4. DEXON Consensus

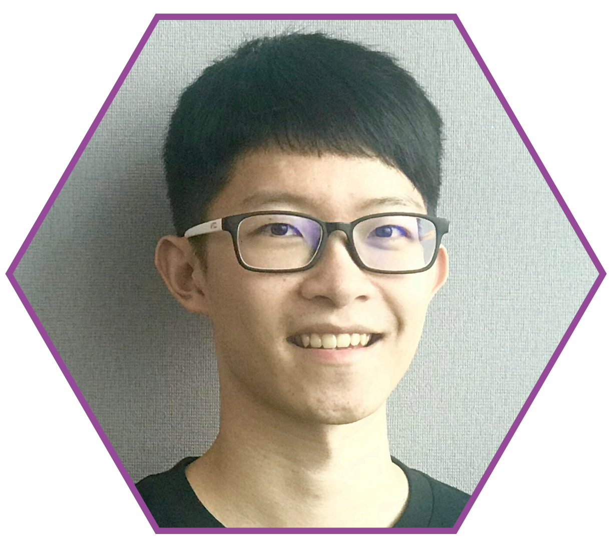
郭博鈞 Po-Chun Kuo

Chief Scientist

- B.S., M.S., PhD program in NTUEE
- Visiting Researcher in TU Darmstadt in Germany, Kyushu University in Japan and University of Haifa, Israel
- Publish 14 papers, 20+ invited talk
- Teaching Assistant 10+ courses in NTU

鍾豪 Hao Chung

Blockchain Researcher



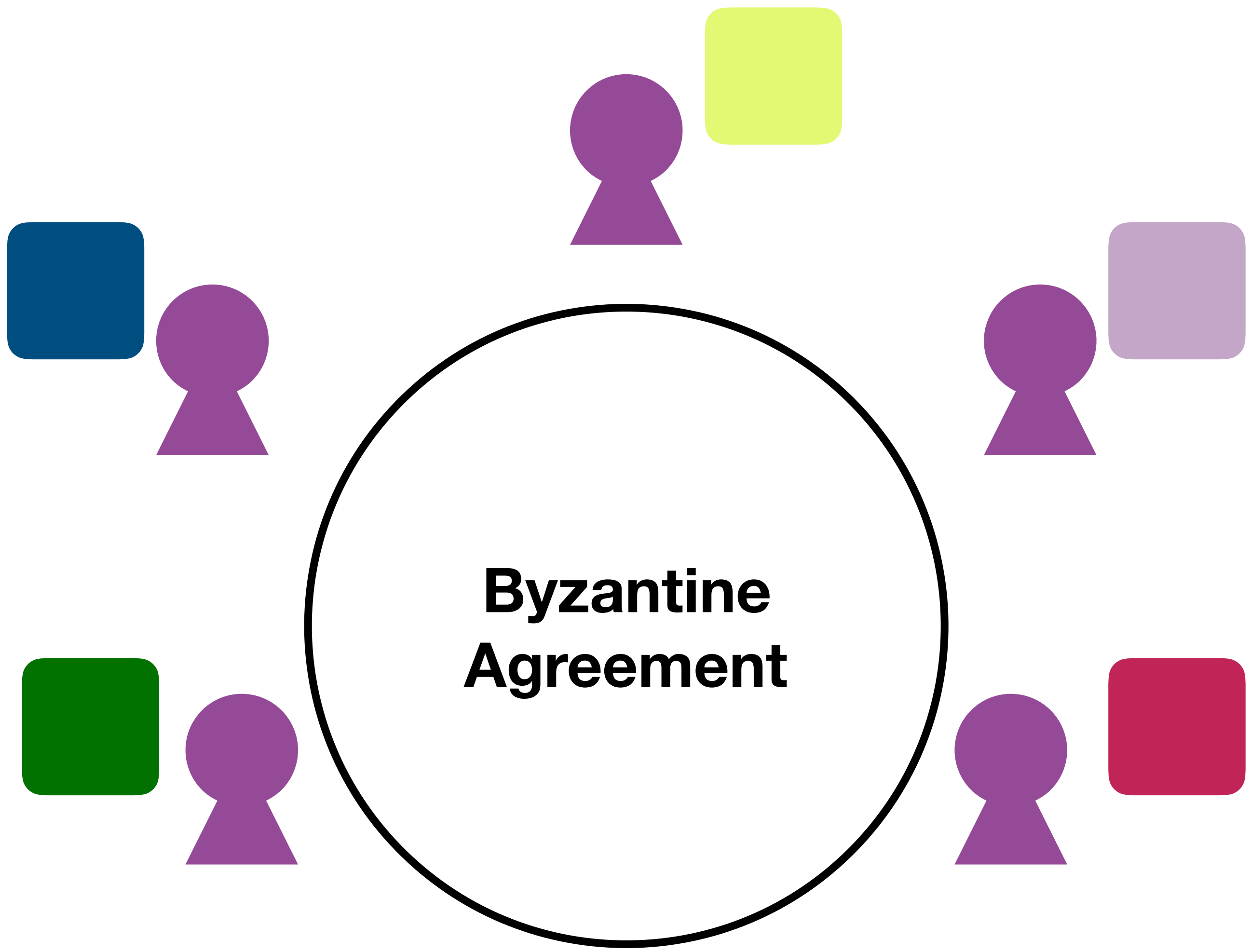
- B.S. in NTU physics, M.S. in NTUEE
- Research in quantum cryptography
- Lecturer and teaching assistant of Crypto camp in sinica

DEXON

Outline

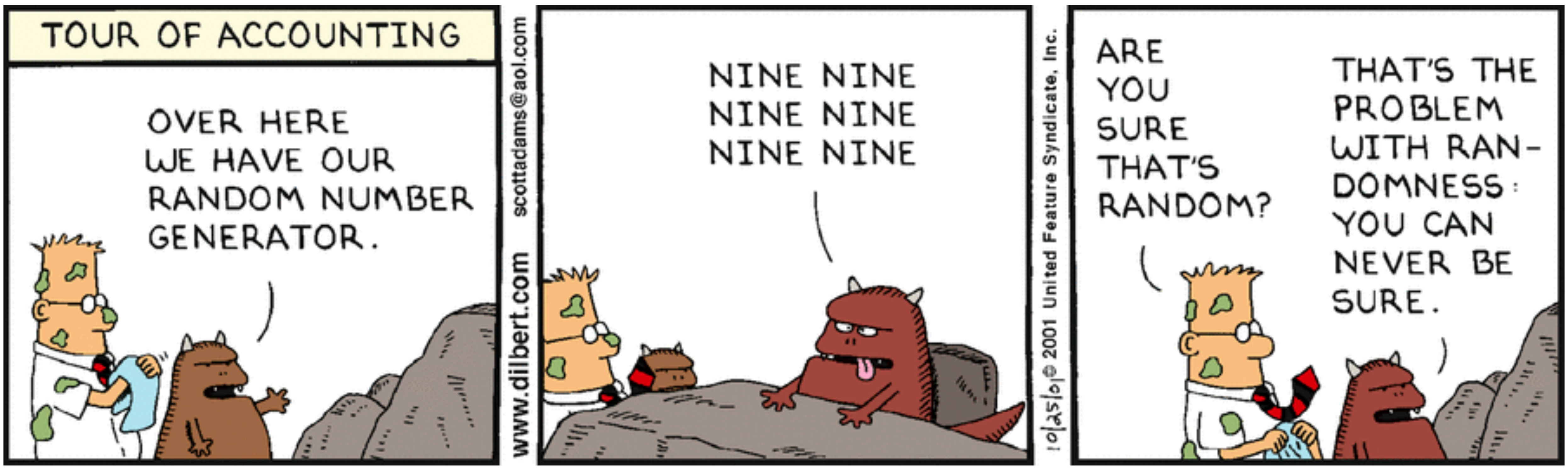
1. Verifiable Random Function (VRF)
2. Threshold Signature
3. Distributed Key Generation (DKG)

In DEXON, we use Byzantine agreement (BA) to help all the miners agree on who can issue the next block.



It would be great that if we can randomly draw one of those nodes.

In DEXON, we use verifiable random function (VRF) to help us.



Can we verify the authentication of the generation of the random number?

Verifiable Random Function

VRF is a function that generates a random value, where the computation can be verified.

$$VRF_{sk}(x) \xrightarrow{(y, \pi)} Veri_{pk}(y, \pi, x) \longrightarrow yes/no$$

Uniqueness

Given a secret key sk and a seed x, a unique output y is generated.

Verification

Given a public key pk, a seed x and a proof π, only one value y can pass the verification.

Pseudorandomness

The output y should look random.

Verifiable Random Function

Remind that the digital signature has the properties:

1. only the user with the secret key can generate a valid signature
2. everyone with the public key can verify the signature
3. without the secret key, the signature should be unpredictable

In DEXON's BA, the VRF is designed as

CRS prevents the users "fit" the secret key beforehand.

an unpredictable seed

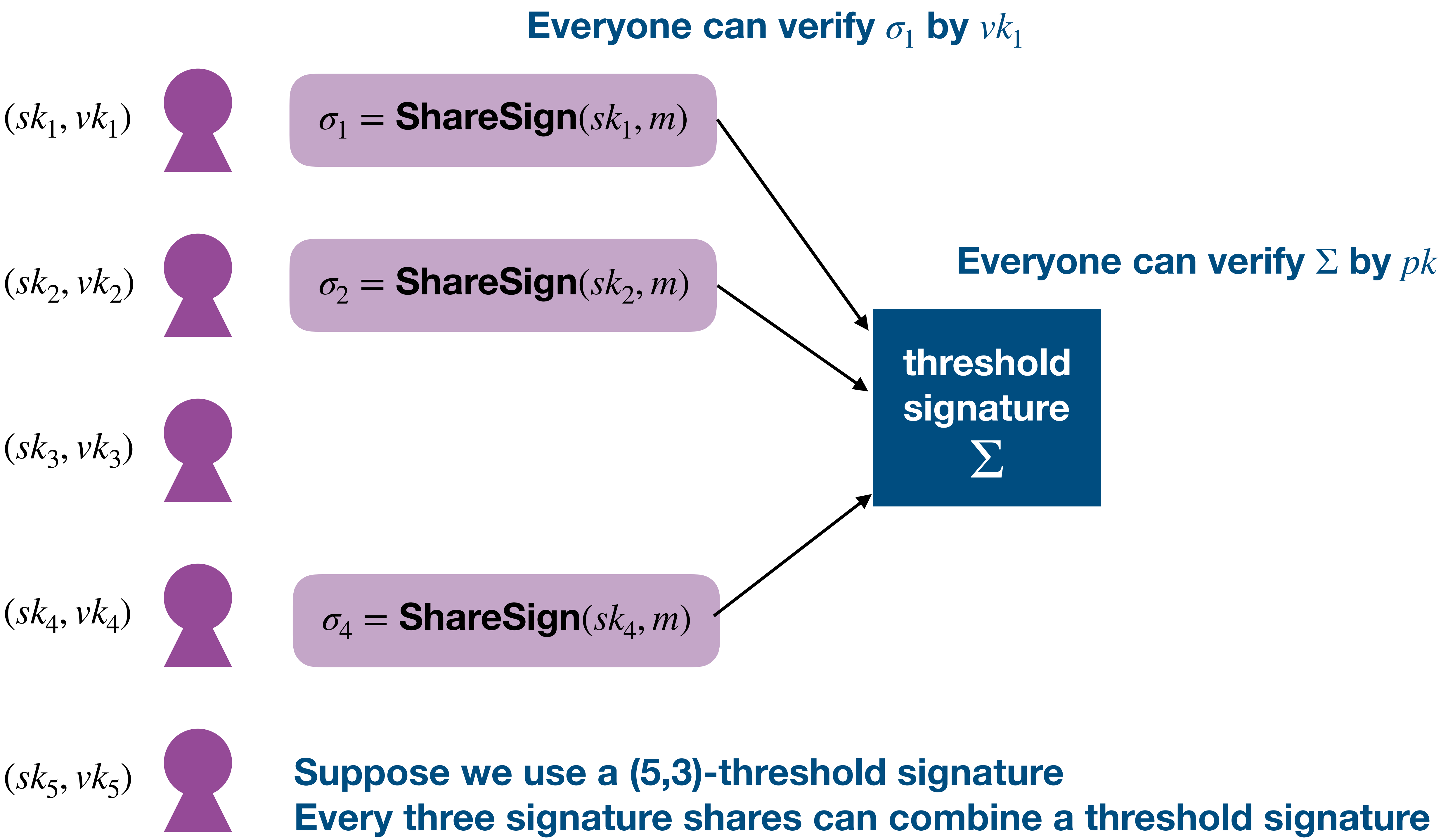
$$\left| R_i - Hash(Sig_{sk}(m)) \right| .$$

make the value pseudorandom

Outline

1. Total Ordering
2. Verifiable Random Function (VRF)
3. Threshold Signature
4. Distributed Key Generation (DKG)

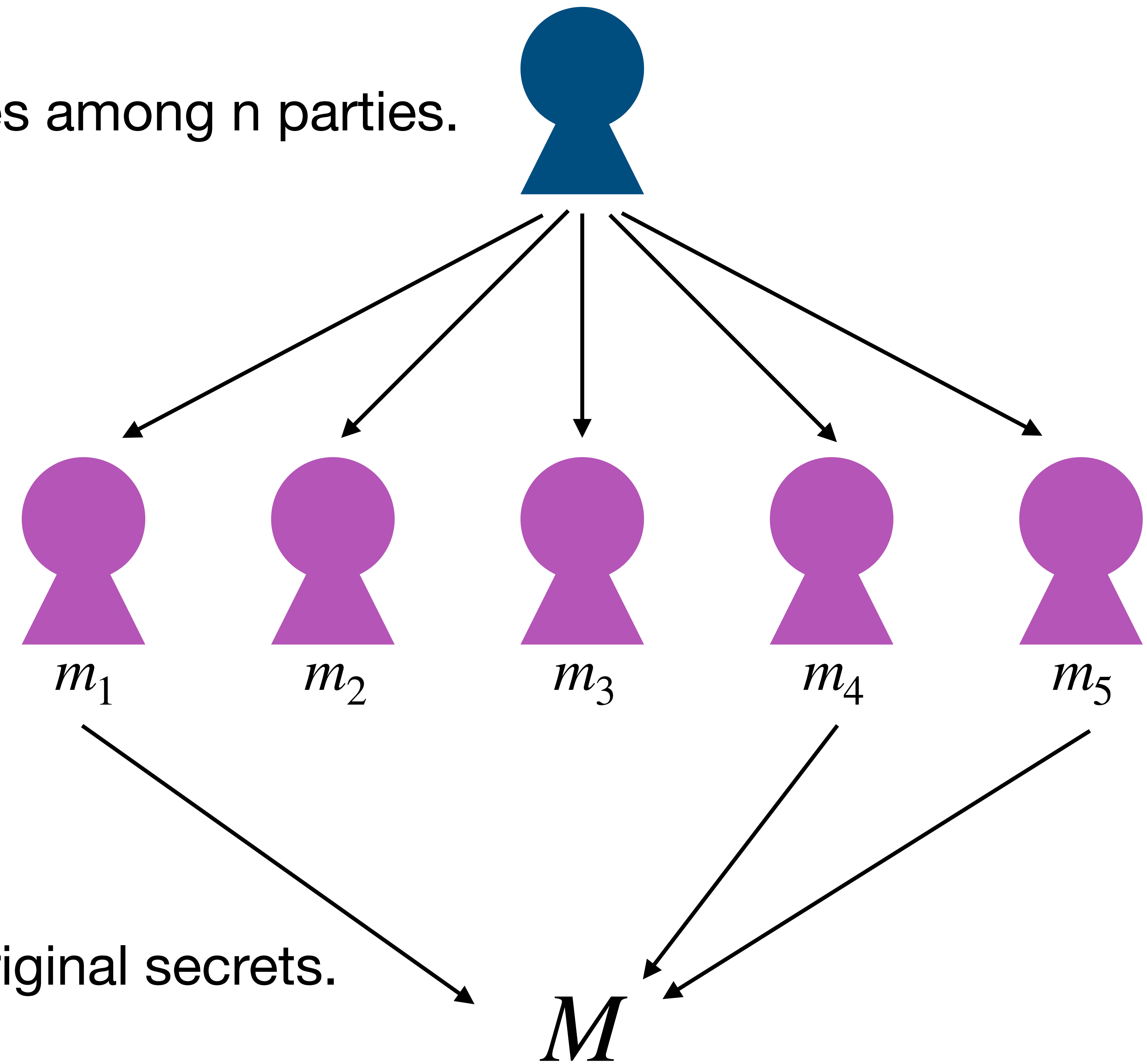
(n,t)-threshold signature



Suppose we use a (5,3)-threshold signature
Every three signature shares can combine a threshold signature

(n,t)-Secret Sharing

A dealer distribute secret shares among n parties.



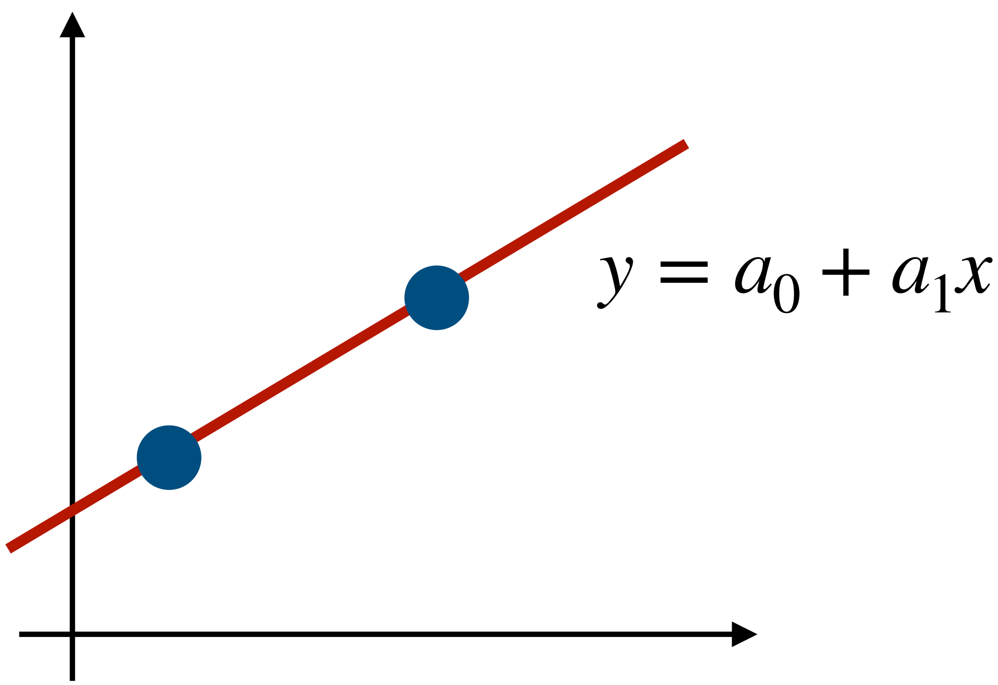
Any t parties can recover the original secrets.

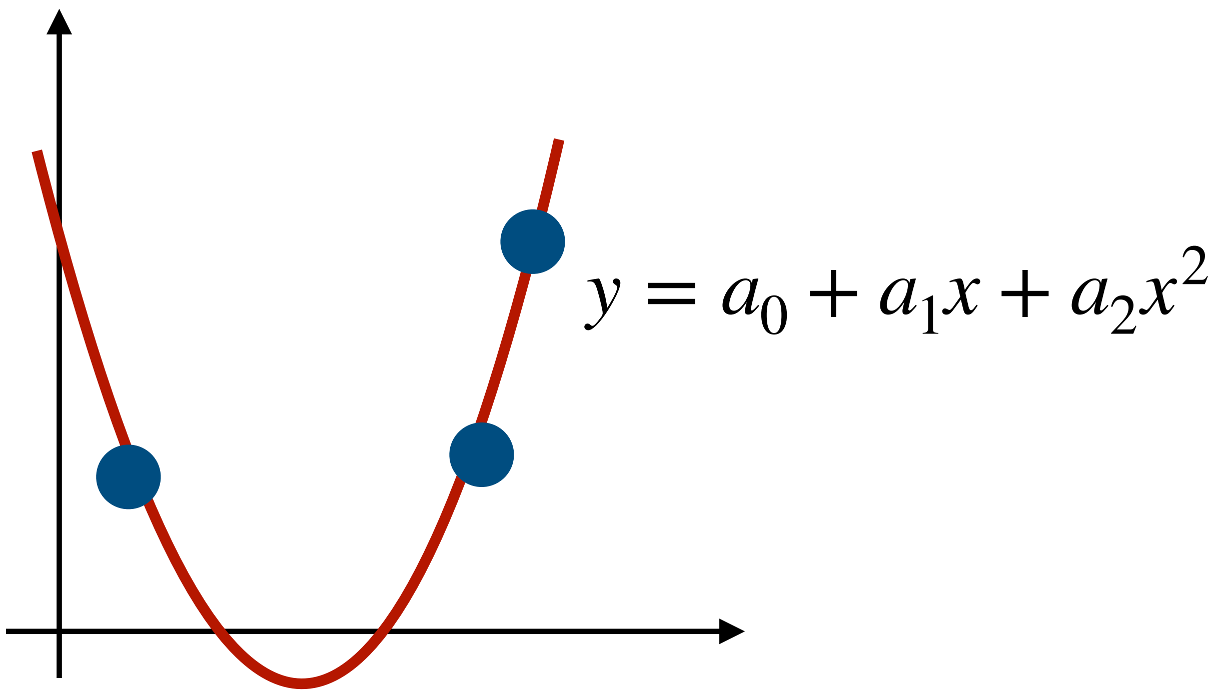
(n,t)-Secret Sharing

An (t-1)-degree polynomial has t parameters:

$$A(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1} .$$

Any t disjoint points can uniquely determine a (t-1)-degree polynomial.





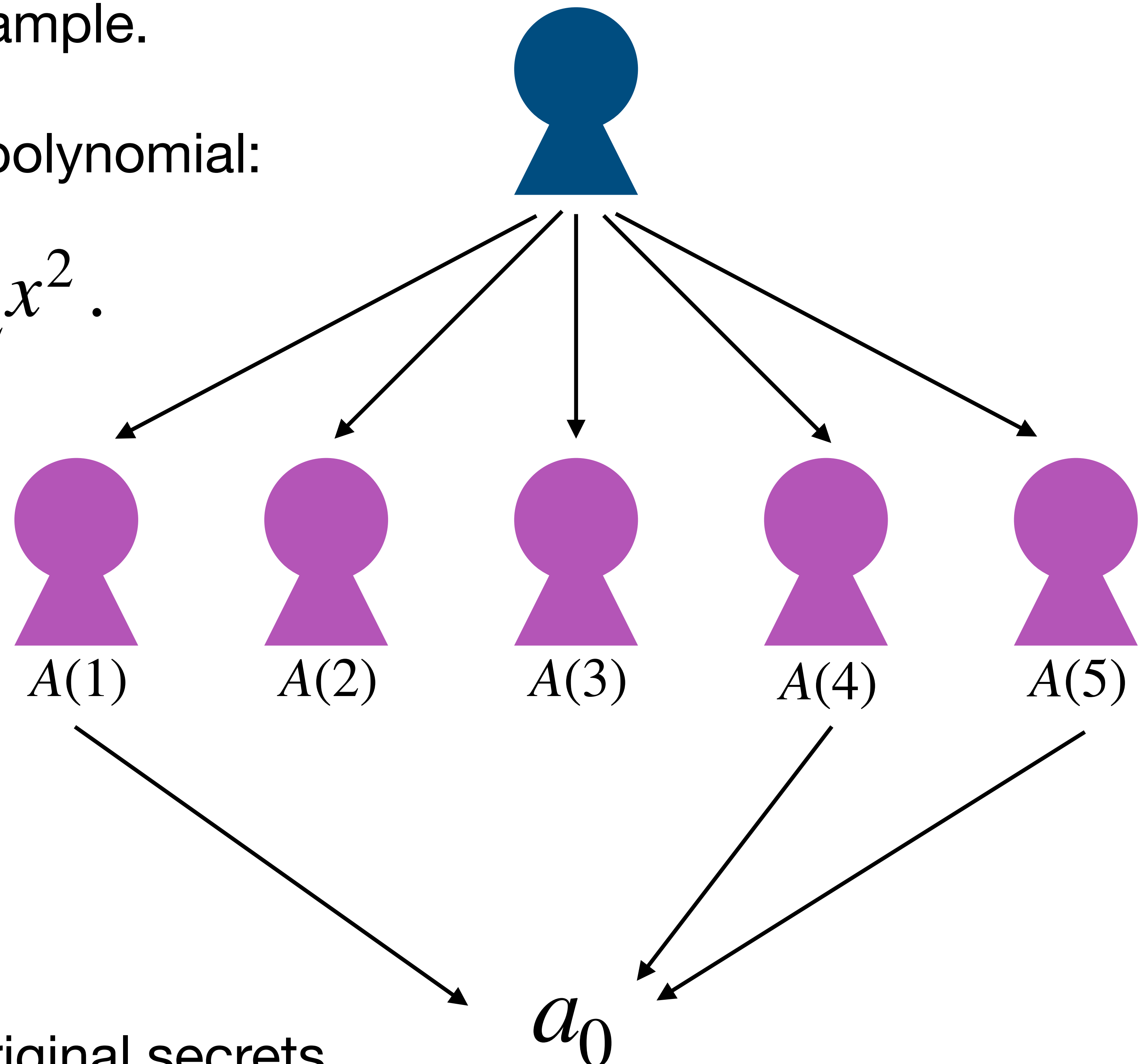
(n,t)-Secret Sharing

Take (5,3)-secret sharing for example.

The dealer choose a 2-degree polynomial:

$$A(x) = a_0 + a_1 x + a_2 x^2 .$$

The secret is hided in a_0 .



$$A(1) = a_0 + a_1 + a_2$$

$$A(4) = a_0 + 4a_1 + 16a_2$$

$$A(5) = a_0 + 5a_1 + 25a_2$$

Any 3 parties can recover the original secrets.

Discrete Logarithm

Give a group $\mathbb{G}$ and a generator g.

$$y = g^x$$

Given g and y, it is difficult to find an x satisfies the equation.

This problem is called discrete logarithm problem.

For example, which x satisfies

$$5 = 3^x \pmod{7}$$

BLS Signature

Pairing

If a function e is pairing, it satisfies

$$e(g_1^x, g_2) = e(g_1, g_2^x)$$

Parameter Setup

Give a hash function $Hash$, a pairing e, and a group generator g.

Let the secret key be x. The public key is g^x.

Signing

Given a message m, the signer computes $h = Hash(m)$.

The signature σ is

$$\sigma = h^x.$$

Verification

The verifier check whether

$$e(\sigma, g) = e(h, g^x)$$

(*n*,*t*)-threshold signature

Suppose an honest dealer prepare a (t-1)-degree polynomial:

$$A(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1} .$$

The dealer distribute $A(i)$ to i-th user.

$$(sk_i, vk_i)$$

$$\|$$

$$(A(i), g^{A(i)})$$

Signing

The i-th user computes

$$h = Hash(m)$$

$$\sigma_i = h^{sk_i}$$

Verification

The verifier checks whether

$$e(\sigma_i, g) = e(h, vk_i)$$

(n,t)-threshold signature

Suppose we have t signature shares.

$$\left.\begin{aligned} \sigma_i &= h^{sk_i} = h^{A(i)} \\ \sigma_{i'} &= h^{sk_{i'}} = h^{A(i')} \\ &\vdots \\ \sigma_{i''} &= h^{sk_{i''}} = h^{A(i'')} \end{aligned}\right\} \text{t shares}$$

We can combine them into a threshold signature:

$$\Sigma = h^{A(0)} .$$

$$h^a \cdot h^b = h^{a+b}$$

$$(h^a)^b = h^{ab}$$

It's exactly what we do in secret sharing, except that now we are in exponent.

(n,t)-threshold signature

Suppose an honest dealer prepare a (t-1)-degree polynomial:

$$A(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1} .$$

The dealer distribute $A(i)$ to i-th user.

The dealer also broadcast $pk = g^{A(0)}$ as the public key.

Notice that the valid signature has a form

$$\Sigma = h^{A(0)} .$$

Everyone can verify the threshold signature by checking:

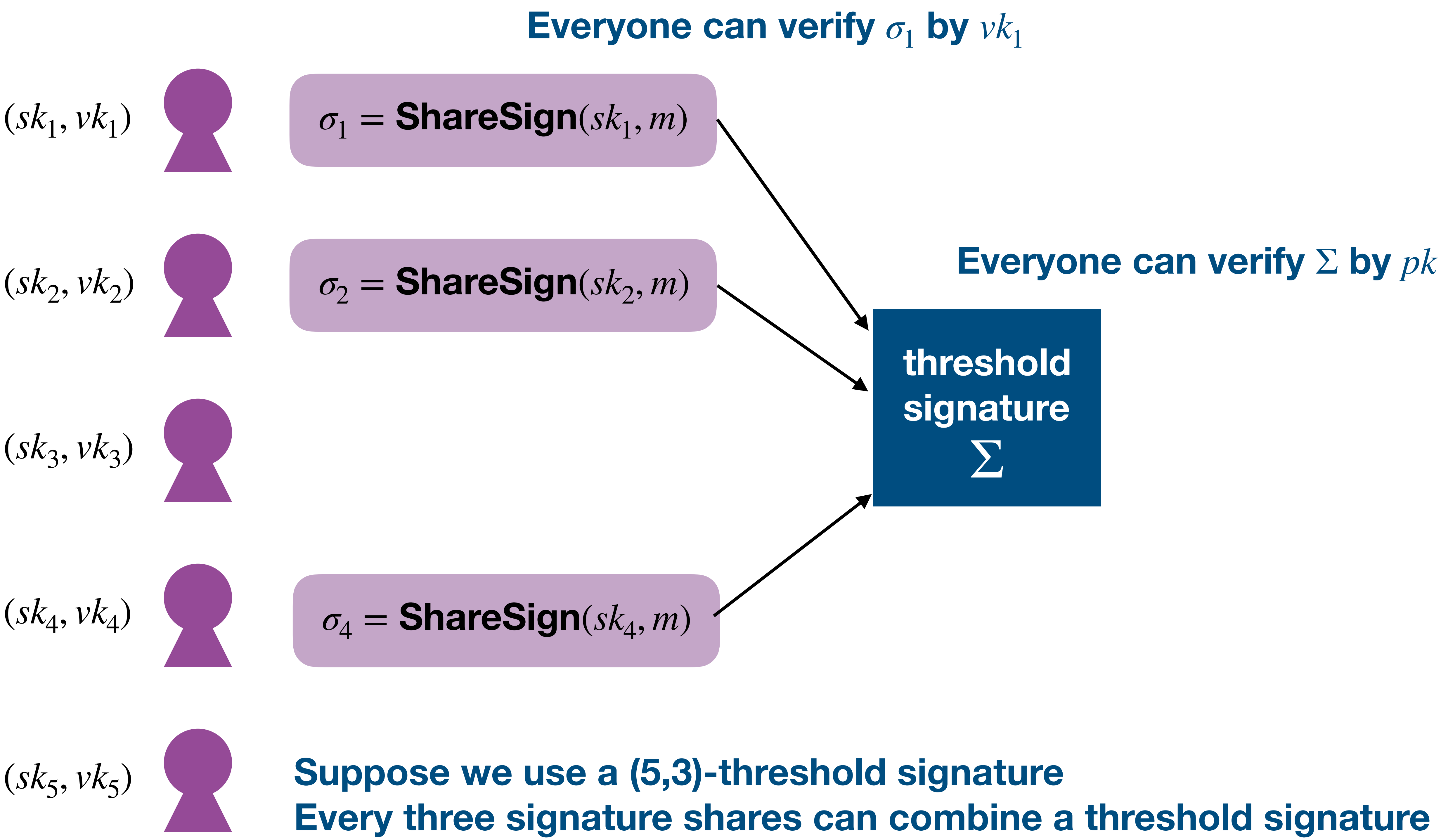
$$e(\Sigma, g) \stackrel{?}{=} e(h, pk) .$$

However, in blockchain, we don’t have an honest dealer.

Outline

1. Total Ordering
2. Verifiable Random Function (VRF)
3. Threshold Signature
4. Distributed Key Generation (DKG)

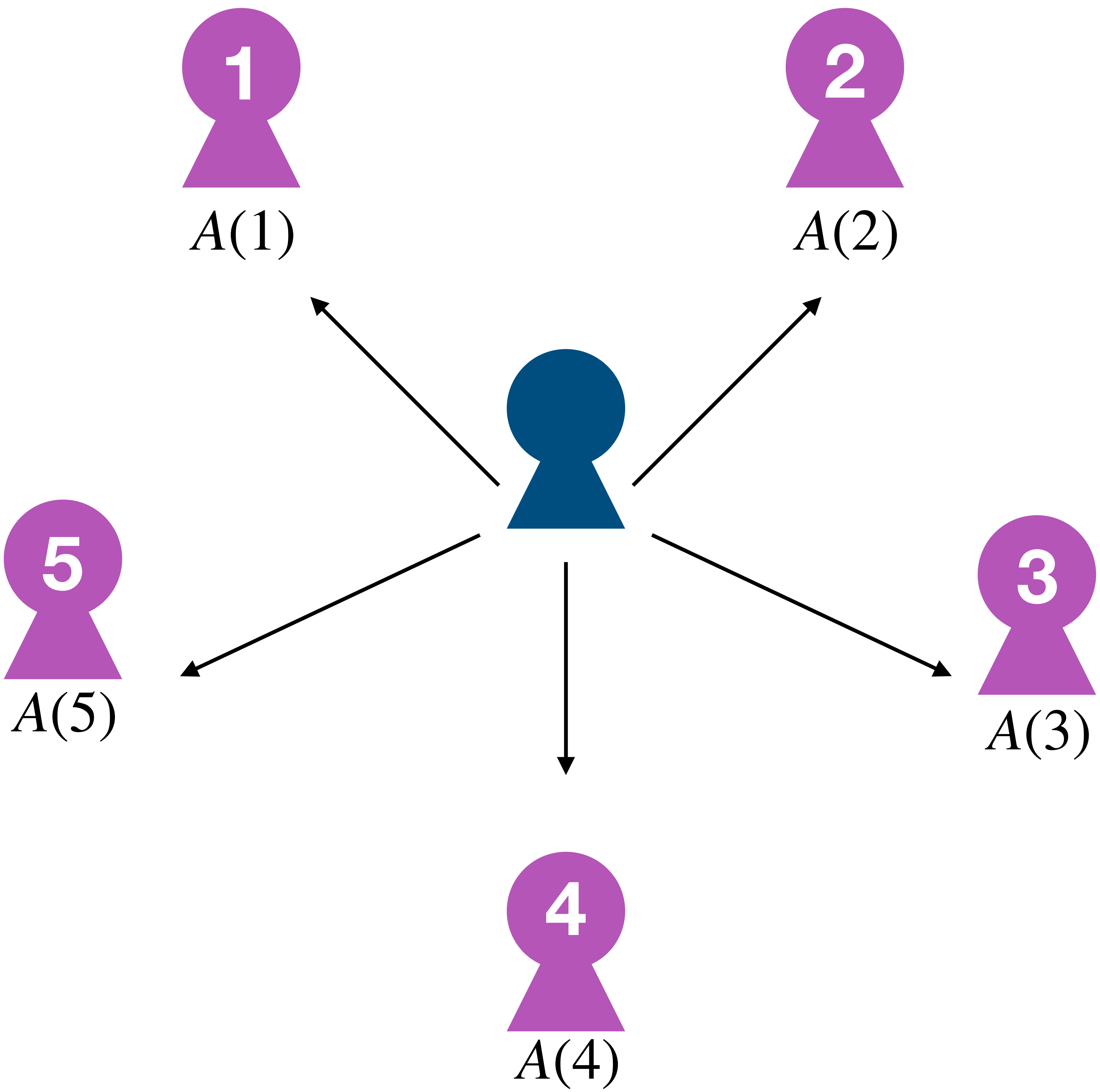
(n,t)-threshold signature



Suppose we use a (5,3)-threshold signature

Every three signature shares can combine a threshold signature

Distributed Key Generation



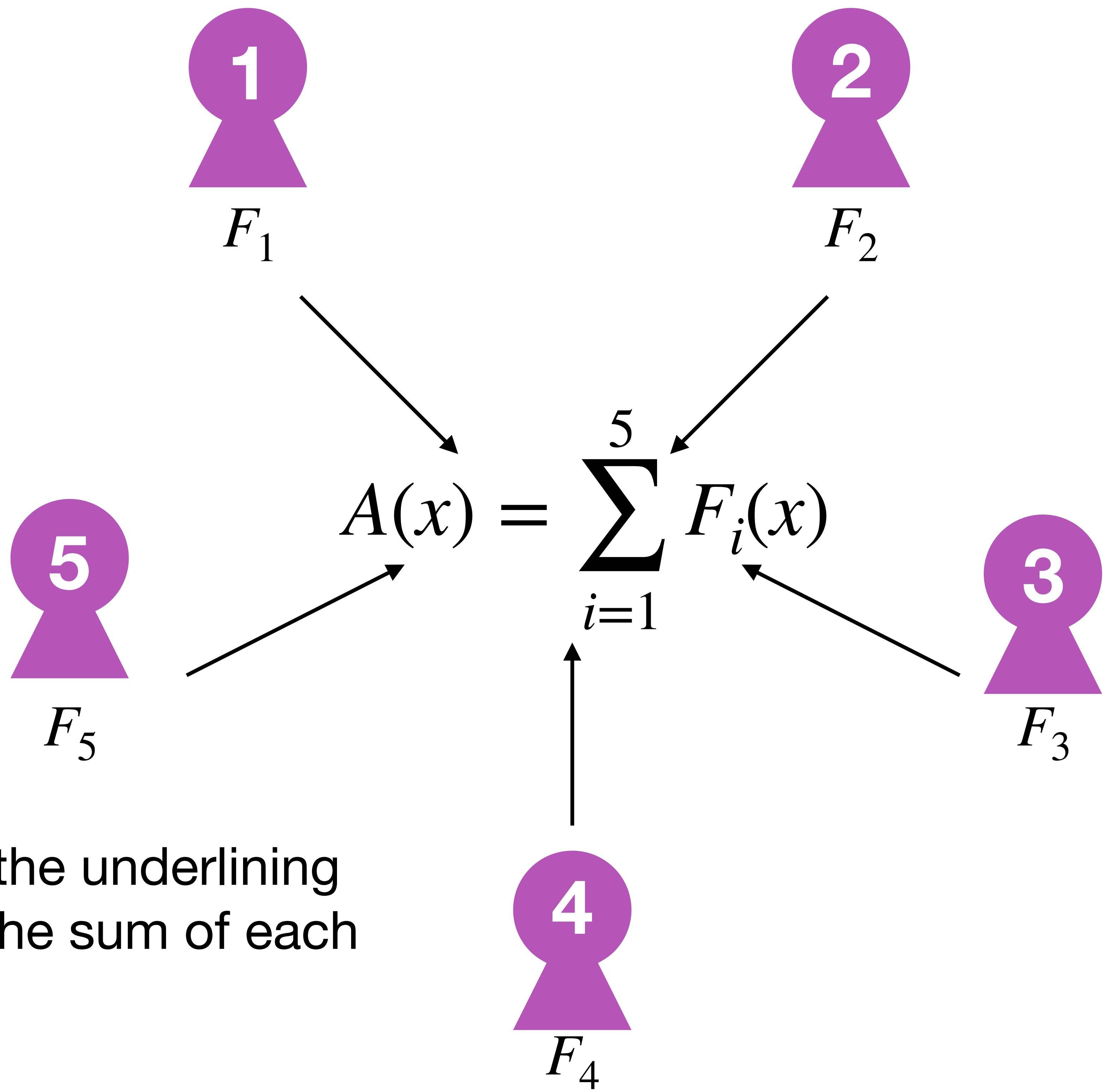
Distributed Key Generation

To remove the honest dealer, each user generates the polynomial individually.

The first user generates a (t-1)-degree polynomial:

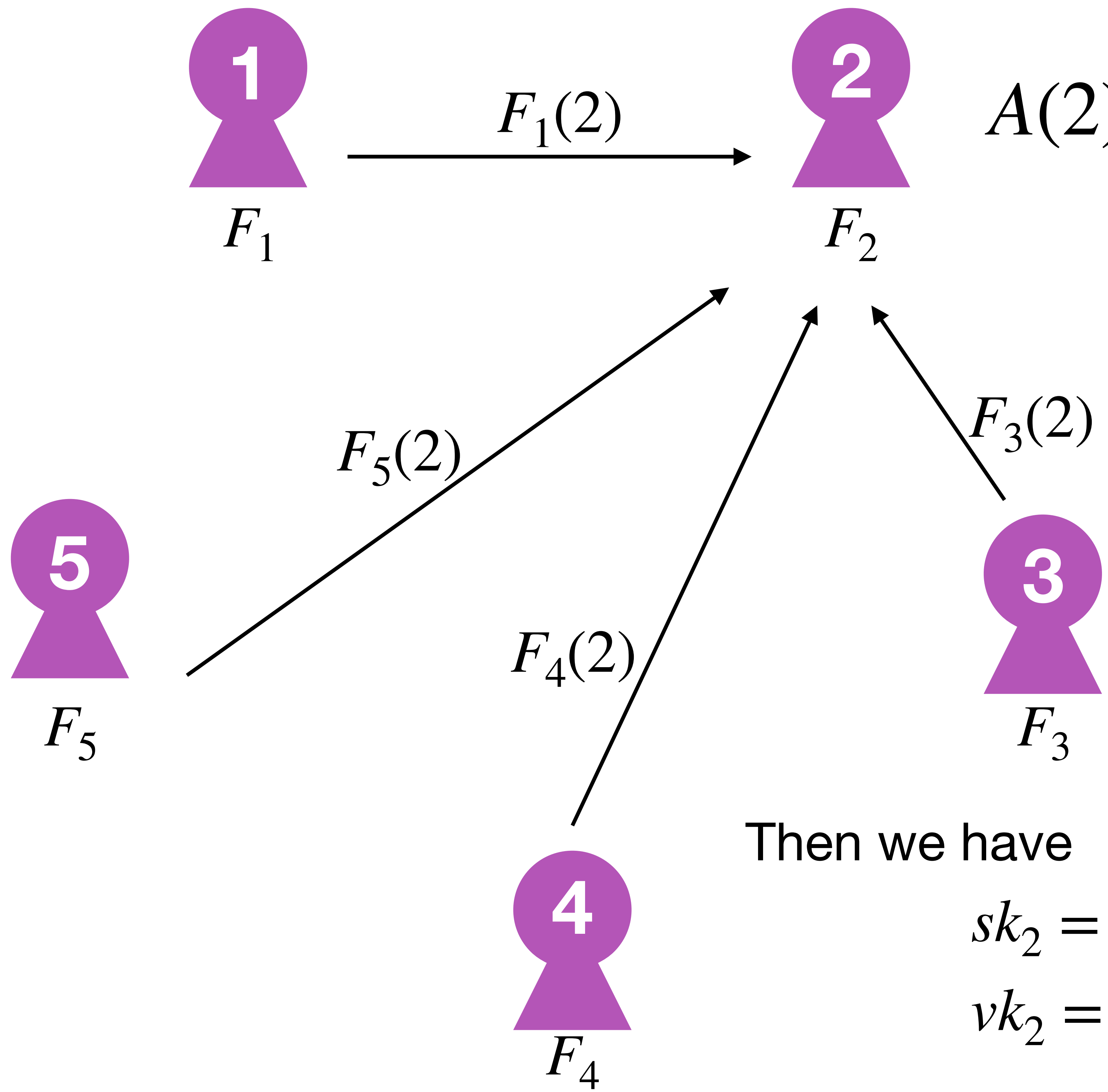
$$F_1(x) = f_{1,0} + f_{1,1}x + \cdots + f_{1,t-1}x^{t-1}.$$

Distributed Key Generation



Conceptually, the underlining polynomial is the sum of each polynomial.

Distributed Key Generation



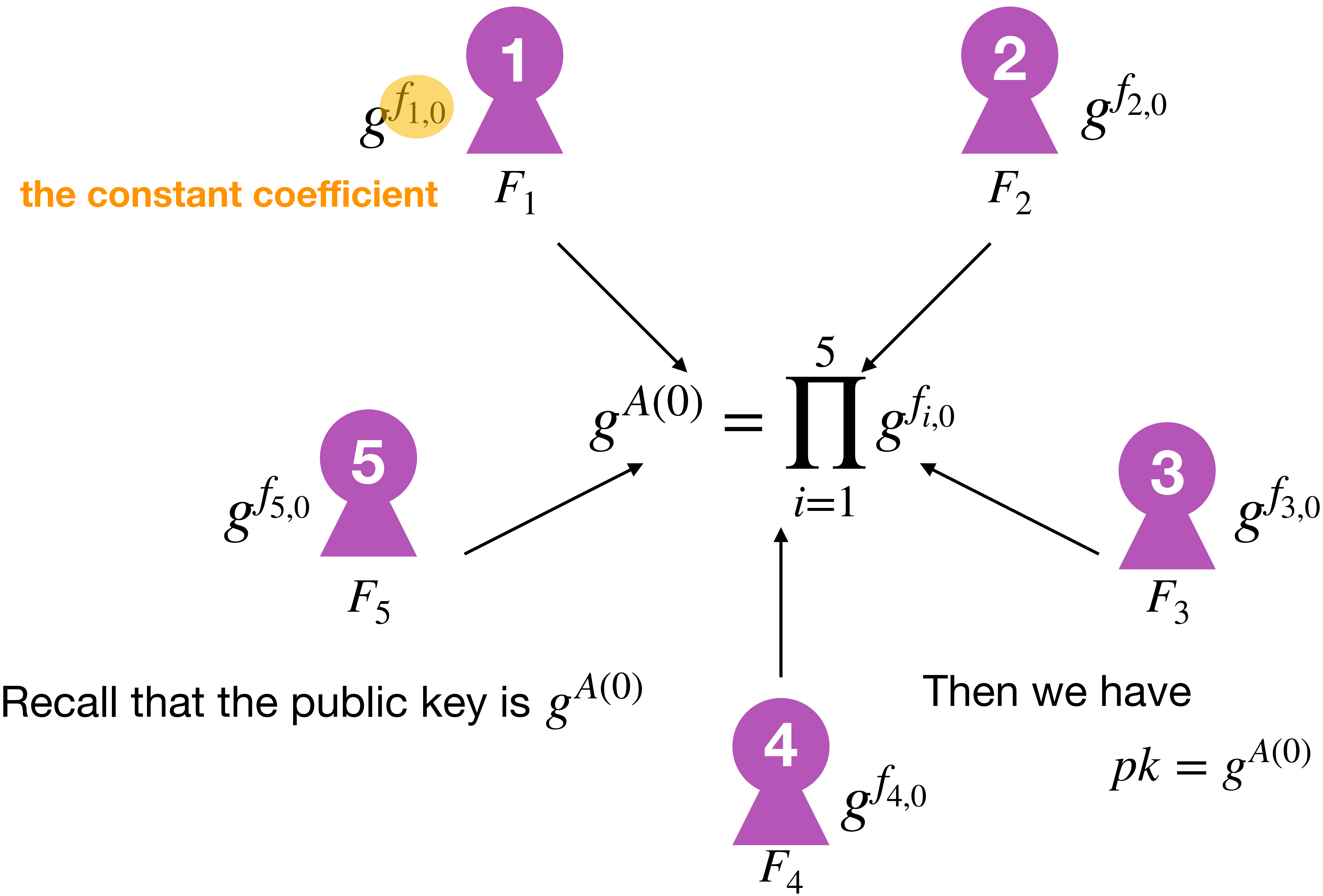
$$A(2) = \sum_{i=1}^{5} F_i(2)$$

Then we have

$$sk_2 = A(2)$$

$$vk_2 = g^{A(2)}$$

Distributed Key Generation



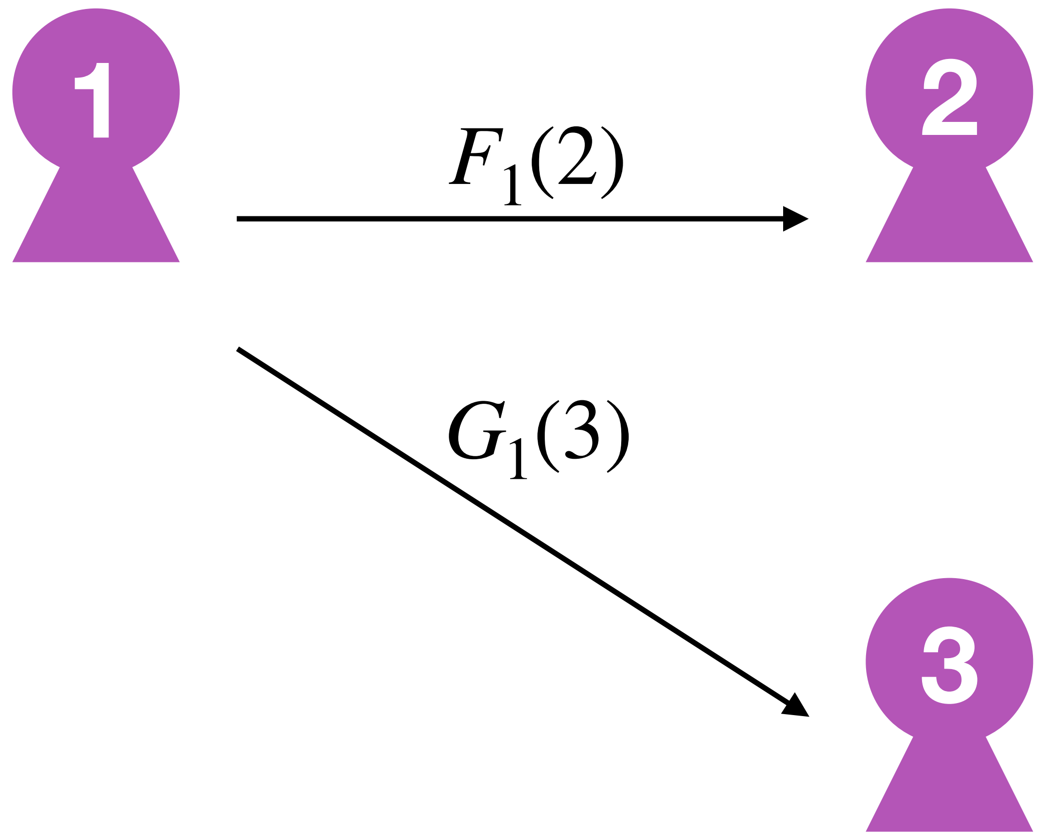
$$g^{A(0)} = \prod_{i=1}^{5} g^{f_{i,0}}$$

Recall that the public key is $g^{A(0)}$

Then we have

$$pk = g^{A(0)}$$

However, if any user equivocates, then the scheme is broken!



How can we resist such Byzantine behavior?

Distributed Key Generation

Suppose the first user generates a (t-1)-degree polynomial:

$$F_1(x) = f_{1,0} + f_{1,1}x + \cdots + f_{1,t-1}x^{t-1}.$$

The first user broadcasts the commitments of all the coefficients.

F_1

$$g^{f_{1,0}}, g^{f_{1,1}}, \cdots, g^{f_{1,t-1}}$$

Other users can check their received value by the commitments.

Take 2nd user for example, he/she just checks

$$g^{F_1(2)} \stackrel{?}{=} g^{f_{1,0}} \times (g^{f_{1,1}})^2 \times (g^{f_{1,2}})^4 \times \cdots \times (g^{f_{1,t-1}})^{2^{t-1}}$$

Distributed Key Generation

For the i-th user:

1. Generates a (t-1)-degree polynomial randomly
$$F_i(x) = f_{i,0} + f_{i,1}x + \cdots + f_{i,t-1}x^{t-1}.$$
2. Broadcast the commitments of the coefficients
$$g^{f_{1,0}}, g^{f_{1,1}}, \ldots, g^{f_{1,t-1}}.$$
3. Send $F_i(j)$ to the j-th user privately
4. Check received messages according to the commitments
5. Compute the keys as follow:
$$sk_i = \sum_{j \in Q} F_j(i)$$
$$vk_i = g^{sk_i}$$
$$pk = \prod_{j \in Q} g^{f_{j,0}}$$

(n,t)-threshold signature