ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Improvements on Distributed Key Generation

Bachelor Project

Kopiga Rasiah

Responsible
Bryan Ford

Supervisor
Nicolas Gailly

Improvements on Distributed Key Generation

- Objective: Bringing improvements in order to enhance the security of the protocol

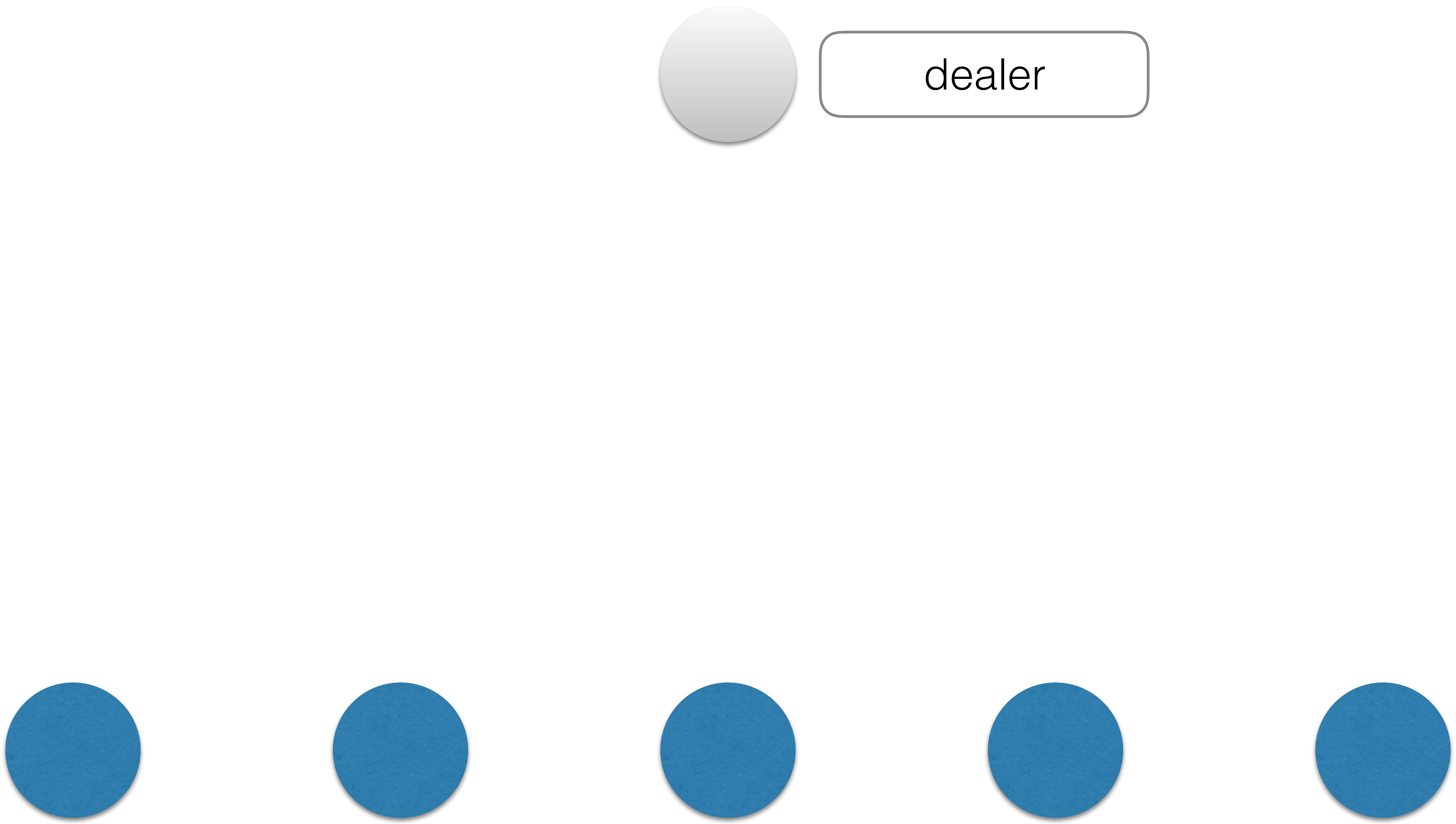
Outline

- Background:
  - What is DKG
  - Shamir’s secret
  - Feldman’s VSS
- How DKG works
- My work: Proactive secret sharing
- Implementation
- Conclusion

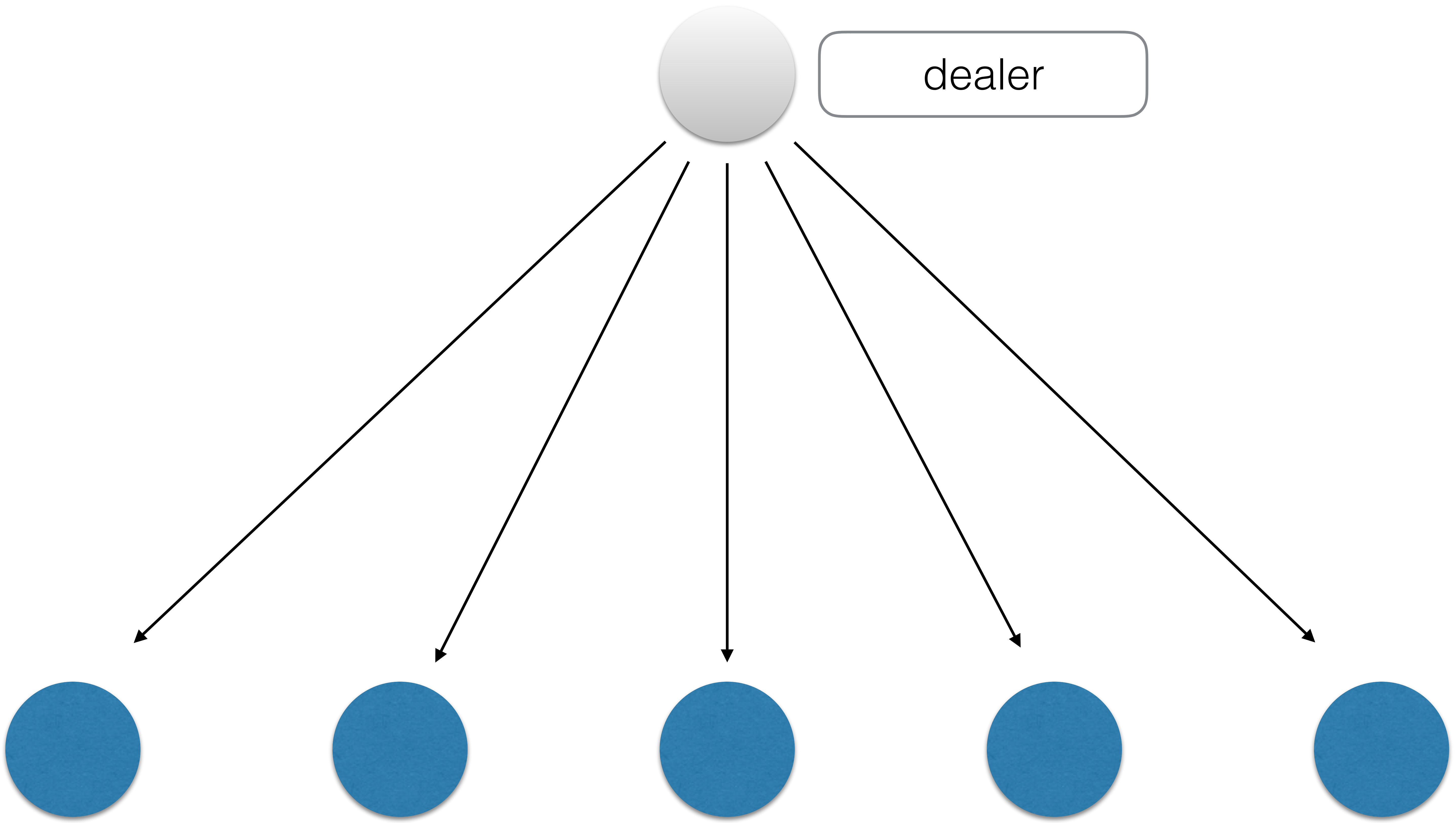
Distributed Key Generation

- Set of *n* participants who collectively generate a shared private/public key
- Each node have a *share* of the secret (private key)
- No single point failure: attacker needs to break into multiple location to have access to the secret.
- DKG is mostly used in group digital signature, or decrypt shared ciphertexts.

Shamir’s secret sharing



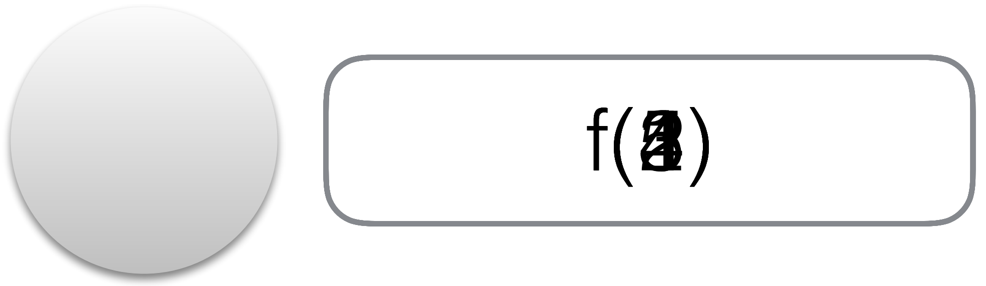
Shamir's secret sharing



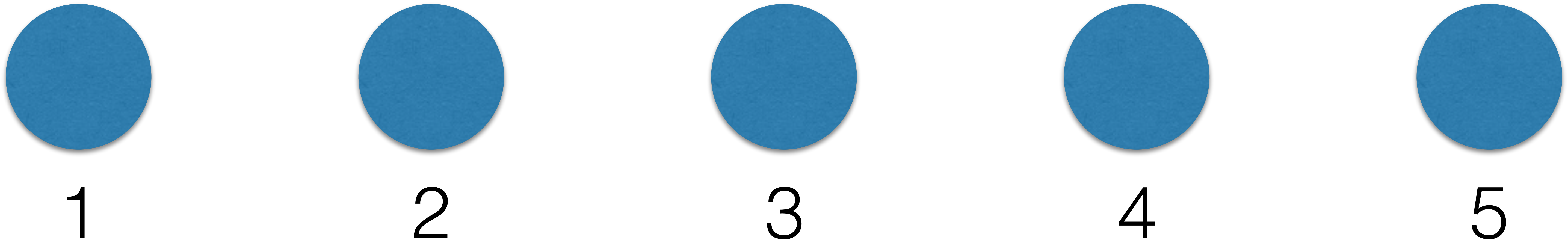
Shamir's secret sharing

- $f(x) = s + a_1x + a_2x^2 + \ldots + a_{t-1}x^{t-1}, \quad t < n$
- **f(0) = secret**
- construct *n* points out of it (shares) and distributes to the nodes

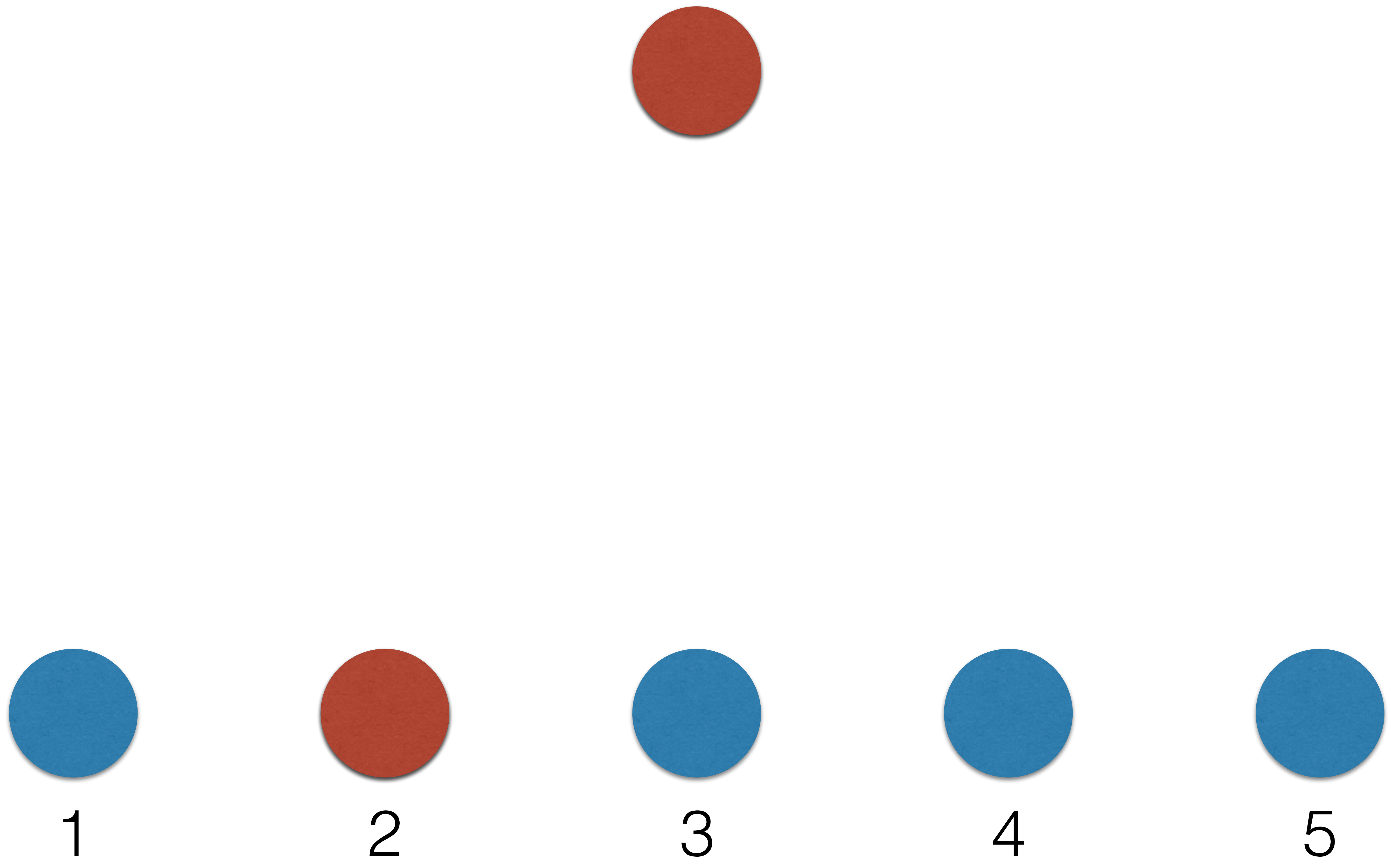
Shamir’s secret sharing



- *t* points are sufficient to reconstruct a *t-1* degree polynomial function



Shamir's secret sharing



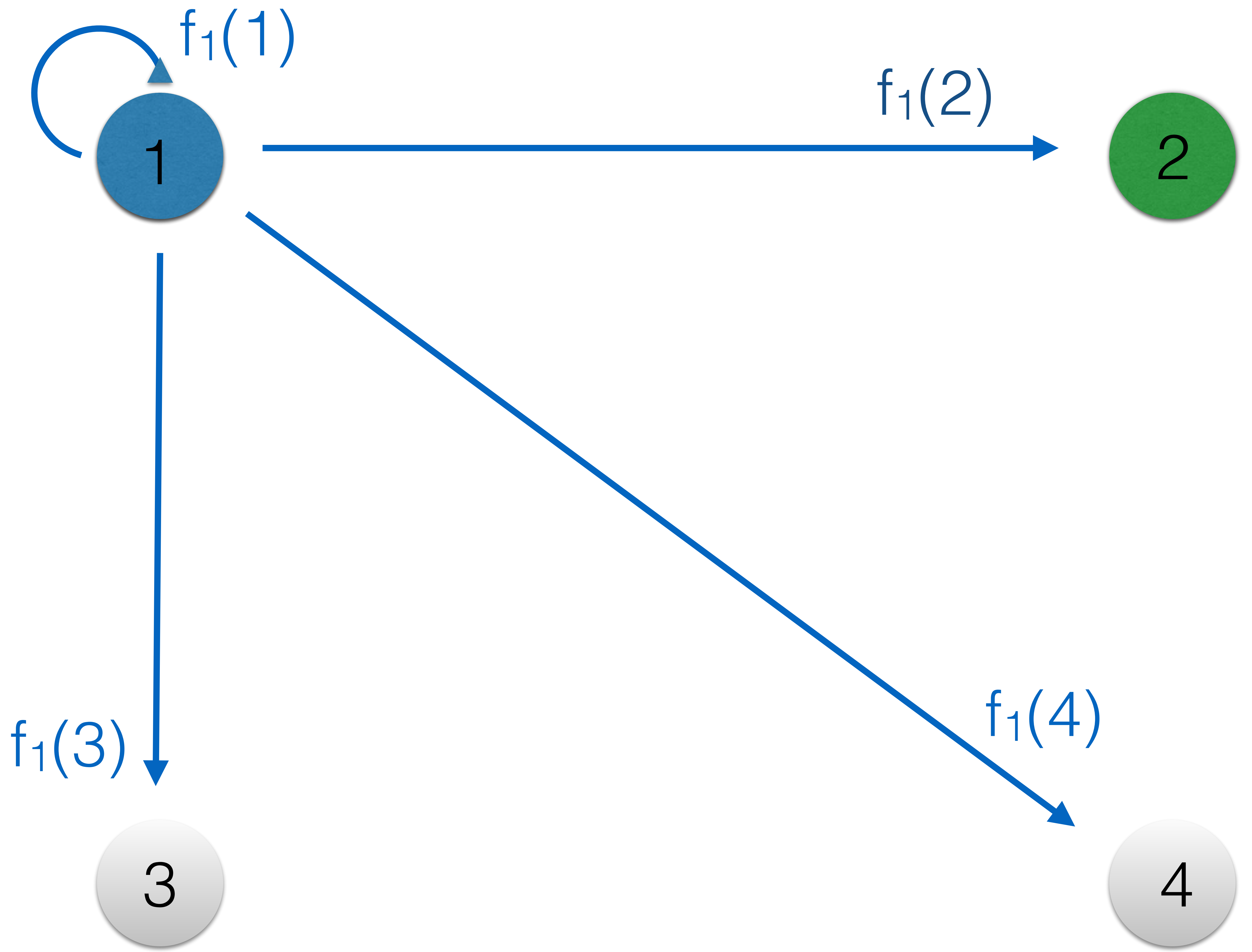
Feldman's verifiable secret sharing

- Based on Shamir's secret sharing
- nodes can verify if their shares are consistent
- dealer broadcasts F(•) = f(•) $*$ g
- $F(i) == s_i * g$

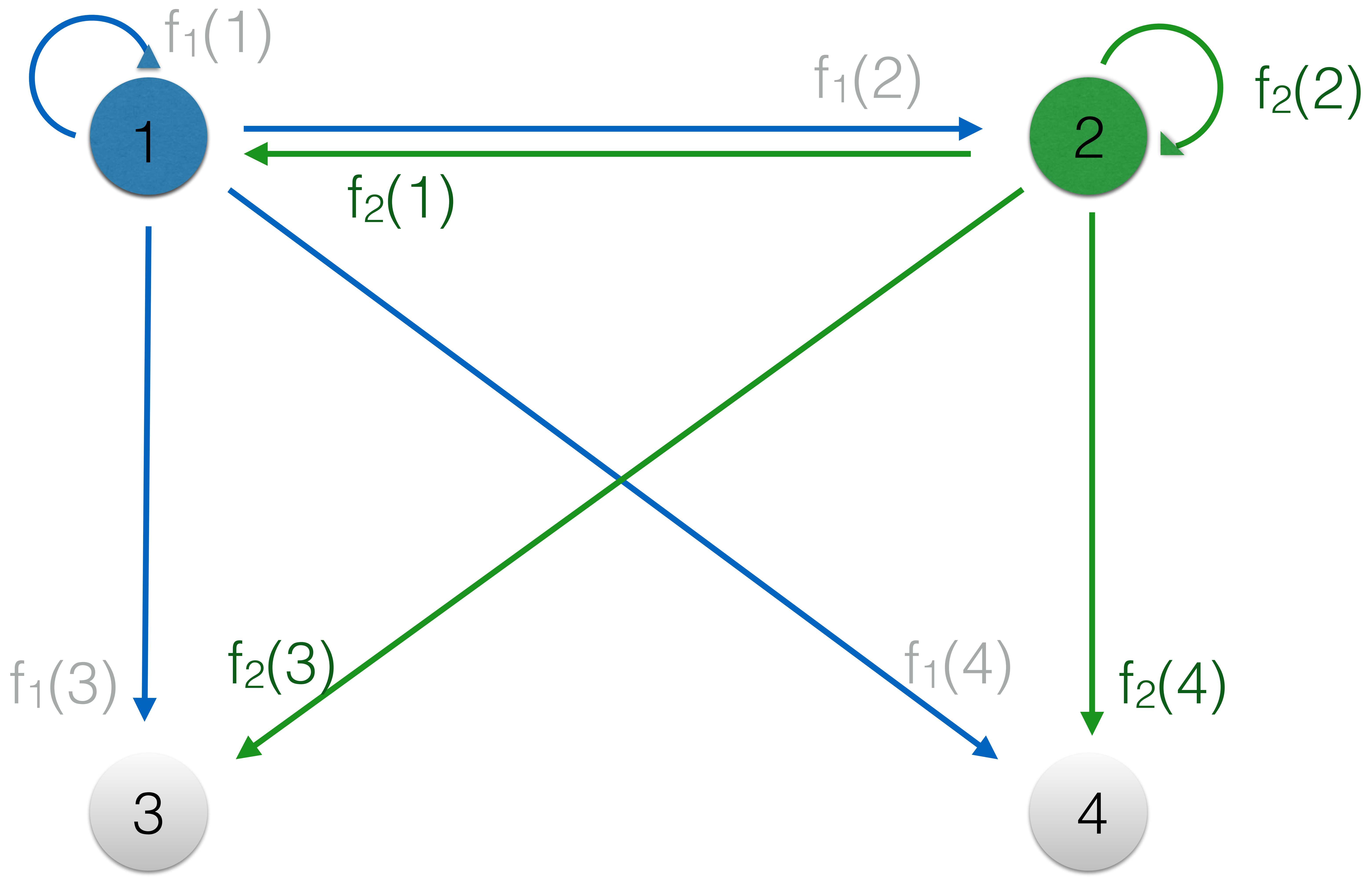
Distributed Key Generation

- Based on Feldman's VSS
- System without any trusted party
- Executes ***n*** VSS instances in parallel: every node is a dealer
- Each node generates $f_i(x) = z_i + a_1x + a_2x^2 + \ldots + a_{t-1}x^{t-1}$, where z_i is random

Distributed Key Generation



Distributed Key Generation



Distributed Key Generation

1

$$f_1(1) + f_2(1) + f_3(1) + f_4(1) = s_1$$

2

$$f_1(2) + f_2(2) + f_3(2) + f_4(2) = s_2$$

3

$$f_1(3) + f_2(3) + f_3(3) + f_4(3) = s_3$$

4

$$f_1(4) + f_2(4) + f_3(4) + f_4(4) = s_4$$

Distributed Key Generation

$$f_1(1) + f_2(1) + f_3(1) + f_4(1) = s_1$$

1

$$f_1(2) + f_2(2) + f_3(2) + f_4(2) = s_2$$

2

$$s = \sum_j f_j(0)$$

$$S = \sum_j F_j(0) = s * g$$

$$f_1(3) + f_2(3) + f_3(3) + f_4(3) = s_3$$

3

$$f_1(4) + f_2(4) + f_3(4) + f_4(4) = s_4$$

4

Proactive secret sharing

- Given enough time, an attacker can gradually break into more than t servers
- Not practical to change the secret
- Solution: Proactive secret sharing.
- We only focus on refreshing the shares

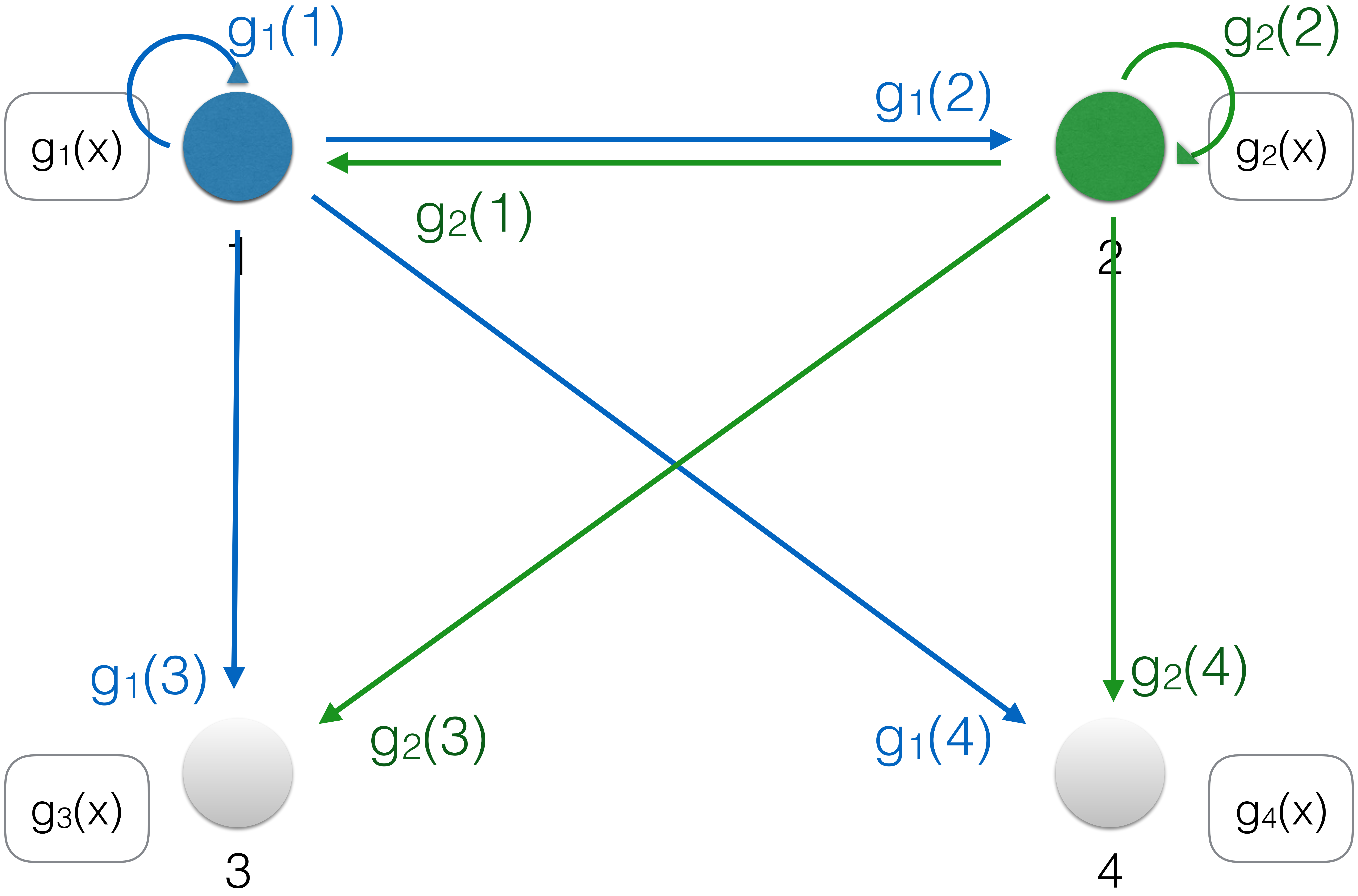
Proactive secret sharing

- Why refreshing ?
- Refreshing the shares makes the underlying polynomial change !
- Old stolen information become useless

The idea

- Let's assume that the initial DKG round has been done
- Each node generates new intermediate random polynomials $\mathbf{g_i(x)}$
- $g_i(x) = 0 + b_{1,i}x + b_{2,i}x^2 + \ldots + b_{t-1,i}x^{t-1}$
- They execute again the DKG protocol:
- distributions of the **intermediate shares**

Distributed Key Generation



Proactive secret sharing

$s_i = \sum_j f_j(i)$ for node i

$s_i' = \sum_j g_j(i)$

Proactive secret sharing

$$
\begin{array}{rl}
 & s_i = \sum_j f_j(i) \\
+ & s_i' = \sum_j g_j(i) \quad \leftarrow \text{2}^{\text{nd}} \text{ round DKG} \\
\hline
 & r_i = \sum_j h_j(i)
\end{array}
$$

Proactive secret sharing

$$
\begin{array}{rl}
 & s_i = \sum_j f_j(i) \\
+ & s_i' = \sum_j g_j(i) \\
\hline
 & r_i = \sum_j h_j(i)
\end{array}
$$

Proactive secret sharing

$$\begin{array}{rl} & s_i = \sum_j f_j(i) \\ + & s_i' = \sum_j g_j(i) \\ \hline & r_i = \sum_j h_j(i) \end{array} \qquad \begin{array}{rl} & s = \sum_j f_j(0) \\ + & s_i' = \sum_j g_j(0) \quad = 0 \\ \hline & s = \sum_j h_j(0) \end{array}$$

$$g_i(x) = 0 + b_{1,i}x + b_{2,i}x^2 + \ldots + b_{t-1,i}x^{t-1}$$

Distributed Key Generation

$$\begin{array}{c} s_1 \\ + \sum_j g_j(1) \\ \hline = \; r_1 \end{array}$$

1

$$\begin{array}{c} s_2 \\ + \sum_j g_j(2) \\ \hline = r_2 \end{array}$$

2

$$\begin{array}{c} s_3 \\ + \sum_j g_j(3) \\ \hline = \; r_3 \end{array}$$

3

$$\begin{array}{c} s_4 \\ + \sum_j g_j(4) \\ \hline = \; r_4 \end{array}$$

4

Distributed Key Generation

1

$$\begin{array}{l} s_1 \\ + \sum_j g_j(1) \\ \hline = r_1 \end{array}$$

renewed share

2

$$\begin{array}{l} s_2 \\ + \sum_j g_j(2) \\ \hline = r_2 \end{array}$$

3

$$\begin{array}{l} s_3 \\ + \sum_j g_j(3) \\ \hline = r_3 \end{array}$$

4

$$\begin{array}{l} s_4 \\ + \sum_j g_j(4) \\ \hline = r_4 \end{array}$$

Proactive secret sharing

- The attacker's time is now restricted between the updating process
- He need to break into servers at the same period

Implementation

- 2^{nd} round of DKG for updating the shares:

- Renew function adds 2 shares according to their indices:
  - check if G(0) = 0 (= 0 * g)
  - check share1.index == share2.index

Conclusion

- enhances security of the protocol

- much more interesting if periodicity is implemented

Future work

- Implement the periodicity
- Implement the share recovering process

Current work

- Drand (distributed randomness beacon daemon) where
- nodes collectively produces random values