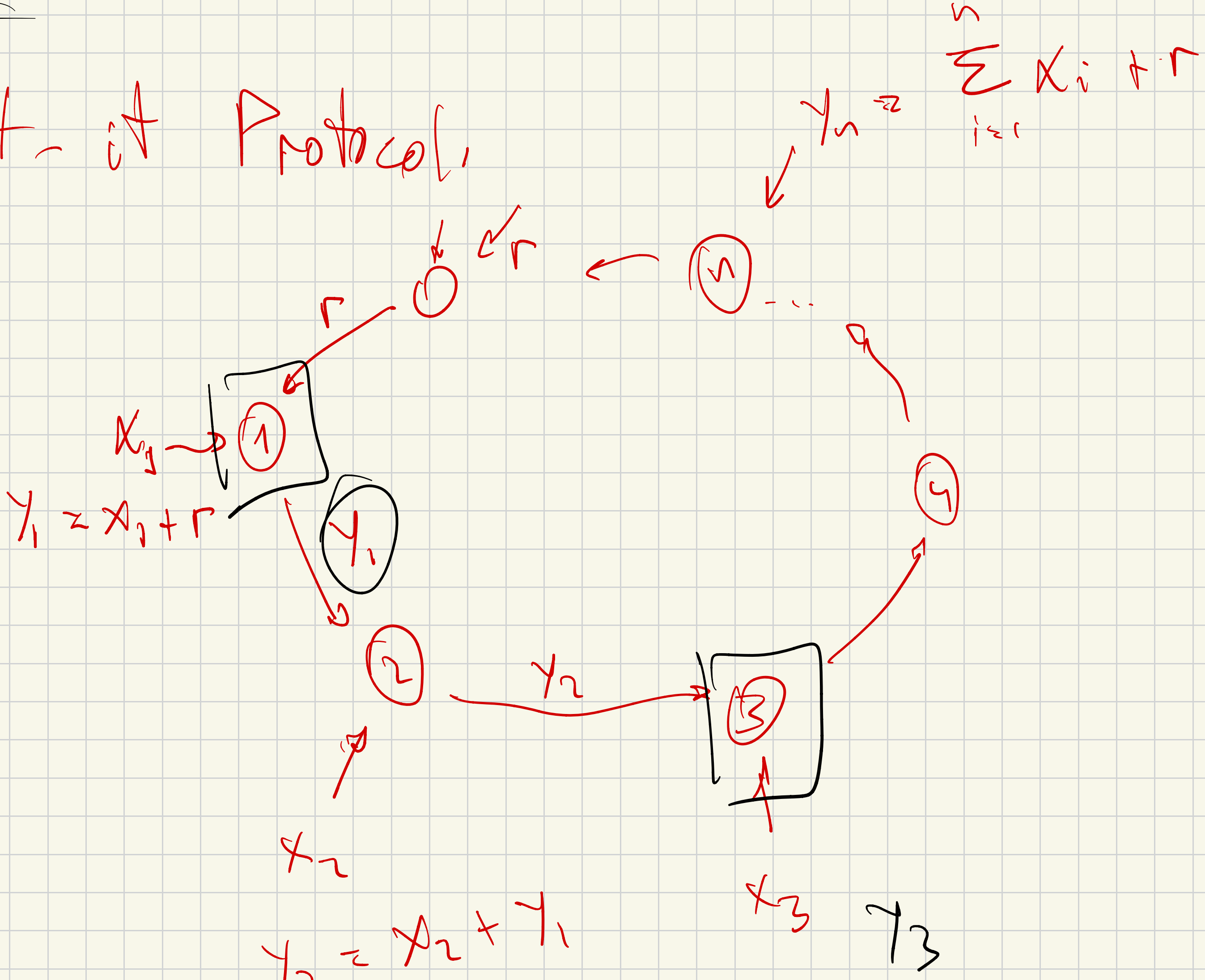
Distributed Key Generation

Chen-Da Liu-Zhang
Research Scientist, Web3 Foundation
Head of Blockchain Lab, Lucerne University of Applied Sciences and Arts

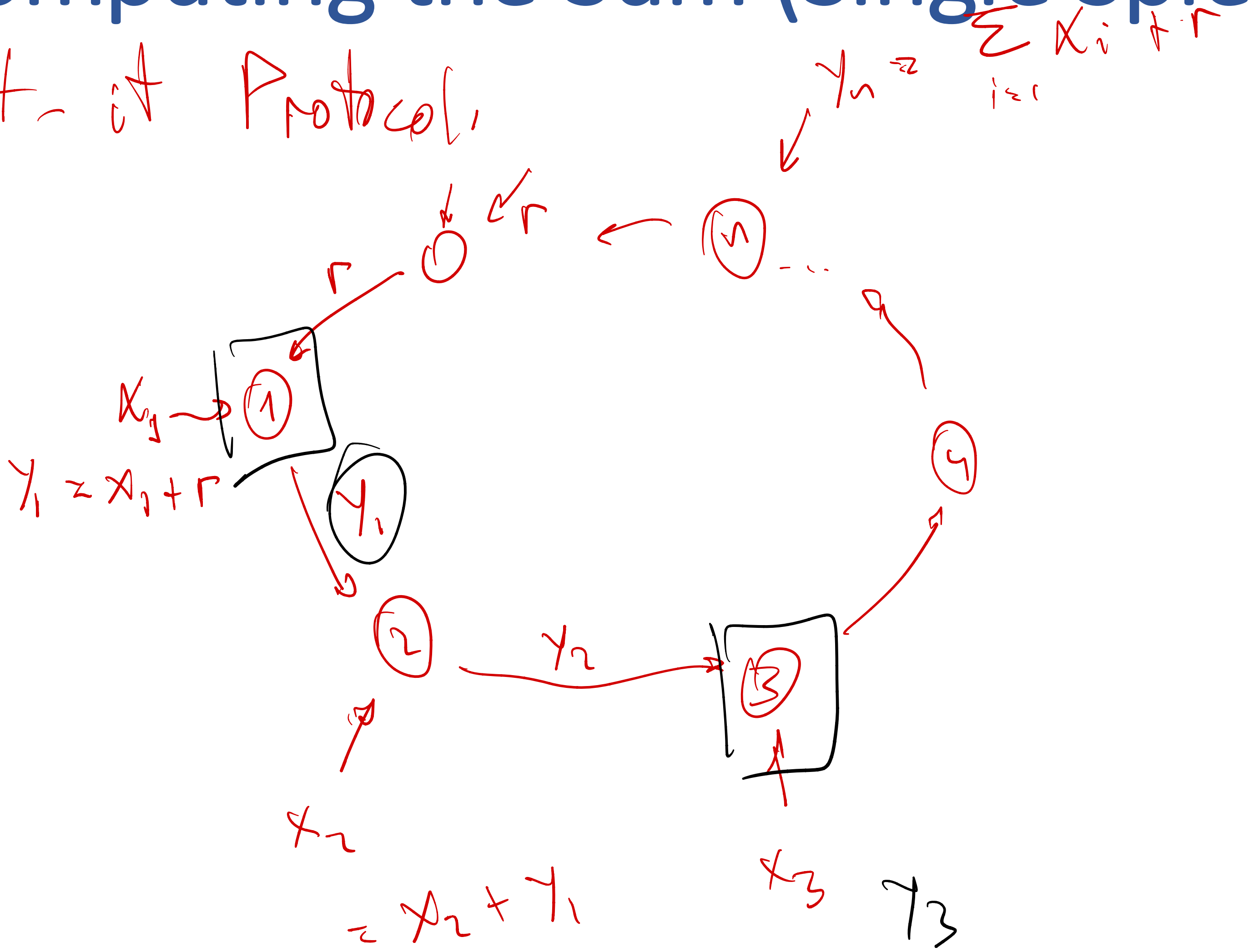
Martin

Post-it Protocol



Computing the Sum (Single Spies)

Post-it Protocol.



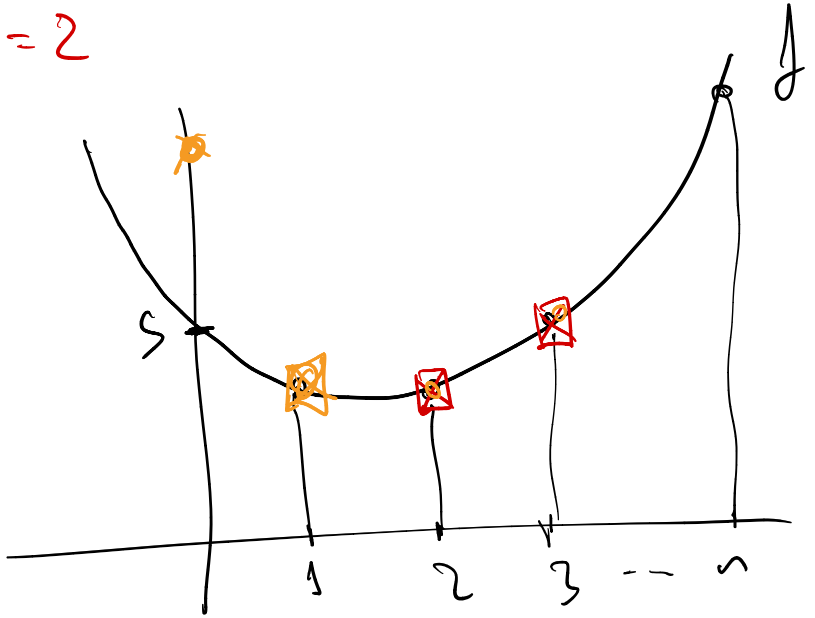
t = 1 single curious guy

t = 2

↳ no privacy.

Shamir Sharing

$t=2$



For general t:

$$f(x) = s + f_1 x + \dots + f_t x^t .$$

Any t have no info on s

Any $t+1$ have full info on s.

Computing the Sum (Threshold of Spies)

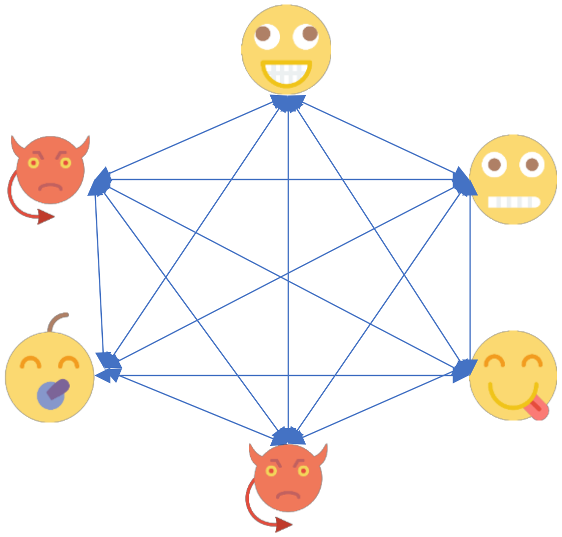
$P_1 \quad x_1 \rightarrow x_{11}, x_{12}, \ldots, x_{1n}$

$\vdots$

$P_n \quad x_n \rightarrow + \, x_{n1}, x_{n2}, \ldots, x_{nn}$

$y_1 \quad y_2 \quad \ldots \quad y_n \quad = \sum_i y_i$

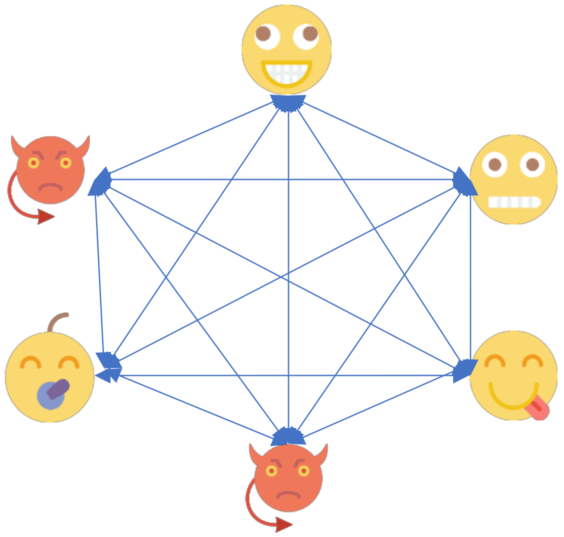
Threshold Cryptography



Setting

- n parties
- t resilience parameter
- Complete network of bilateral channels

Threshold Cryptography



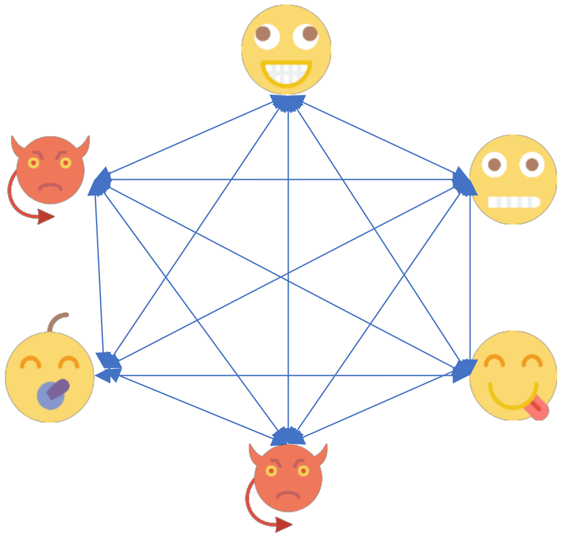
Setting

- n parties
- t resilience parameter
- Complete network of bilateral channels

Goal

- Any $t + 1$ parties can perform some cryptographic operation
- An adversary corrupting t parties cannot

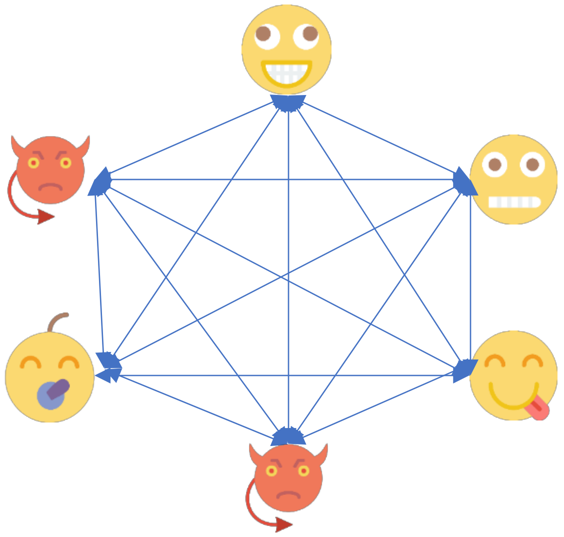
Common Applications



Threshold Signatures

- Any $t + 1$ parties can create a valid **signature**
- An adversary corrupting t parties cannot

Common Applications



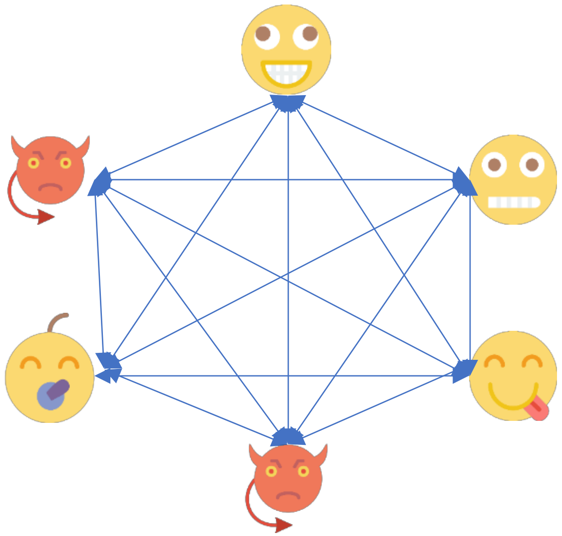
Threshold Signatures

- Any $t+1$ parties can create a valid **signature**
- An adversary corrupting t parties cannot

Threshold Decryption

- Any $t+1$ parties can **decrypt** a ciphertext
- An adversary corrupting t parties cannot

Common Applications



Threshold Signatures

- Any $t + 1$ parties can create a valid **signature**
- An adversary corrupting t parties cannot

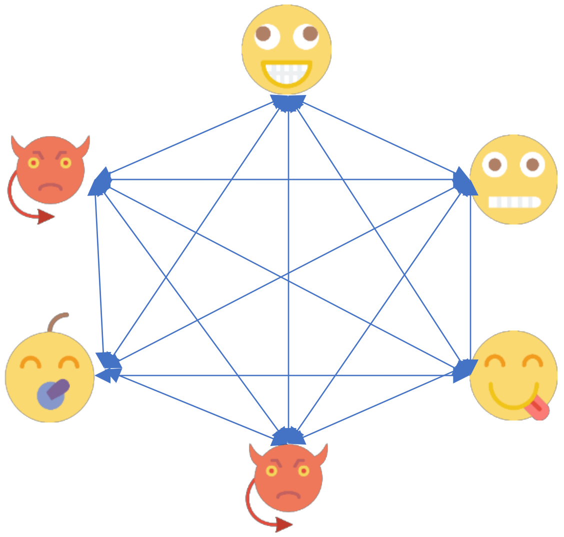
Threshold Decryption

- Any $t + 1$ parties can **decrypt** a ciphertext
- An adversary corrupting t parties cannot

Distributed Randomness Beacons

- Generating unbiased random bits

Common Applications



Threshold Signatures

- Any $t+1$ parties can create a valid **signature**
- An adversary corrupting t parties cannot

Threshold Decryption

- Any $t+1$ parties can **decrypt** a ciphertext
- An adversary corrupting t parties cannot

Distributed Randomness Beacons

- Generating unbiased random bits

Distributed ZK Proofs

- Any $t+1$ parties can generate **proof** on distributed data

Architecture

Distributed Key Generation

Distributed Protocol for the specific task: signing, decrypting, etc

Architecture

Distributed Key Generation

Distributed Protocol for the specific task: signing, decrypting, etc

Distributed Key Generation

Consider the Discrete Logarithm setting:

- n parties
- t corruptions
- G is a cyclic group of prime order q and generator g

Distributed Key Generation

Consider the Discrete Logarithm setting:

- n parties
- t corruptions
- G is a cyclic group of prime order q and generator g

Goal: Distributed protocol for n parties that generate

- Common **public key** $\boldsymbol{y = g^x}$
- The **secret** $\boldsymbol{x}$ is Shamir-shared (with degree-t) across the parties:
  - Each $\boldsymbol{P_i}$ **has share** $\boldsymbol{s_i}$, and $(s_1, \dots, s_n)$ are points on degree- t polynomial
- Common commitment values $(g^{s_1}, \dots, g^{s_n})$

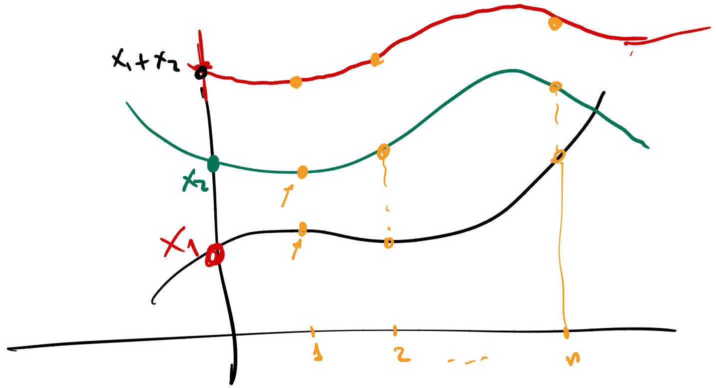
Distributed Key Generation

Properties:

of x (secret key)

- **Correctness**: Each honest P_i obtains a share s_i, and honest shares lie on degree-t polynomial. Moreover, everyone has commitment values $(g^{s_1}, \ldots, g^{s_n})$ and $y = g^x$
- **Secrecy**: Corrupted parties do not learn x (beyond what is leaked by $y = g^x$)
- **Unbiasable**: The public key y is uniformly random

Distributed Key Generation



Sketch:

Parties $P_1, \dots, P_n$ jointly generate a random value as the secret key x

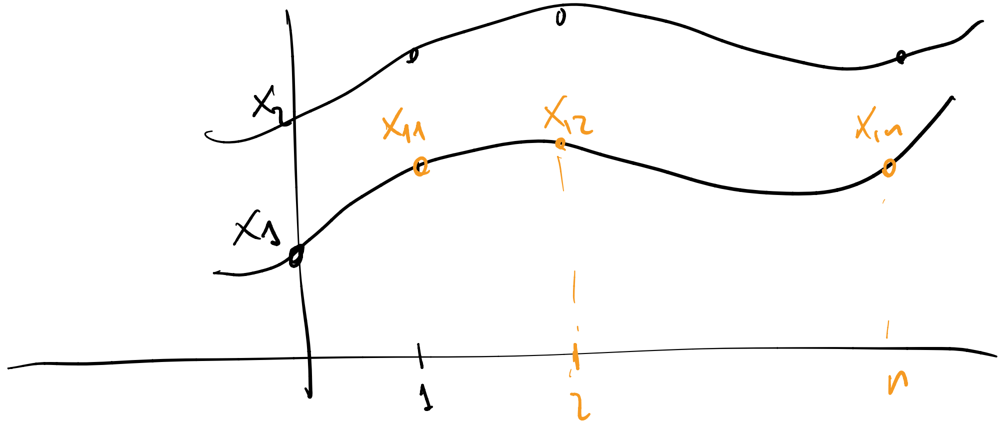
1. Each P_i Shamir-shares a random value x_i

   Every P_j obtains a share $[x_i]_j$ (from each P_i)

2. Each P_j computes the sum of obtained shares $s_j = \sum_i [x_i]_j$

Note that here the values s_j lie on a degree-t polynomial, with secret x

Distributed Key Generation



Sketch:

Parties $P_1, \dots, P_n$ jointly generate a random value as the secret key x

$[g^{x_{11}}, g^{x_{12}}, \dots, g^{x_{1n}}]$

1. Each P_i Shamir-shares a random value x_i and publishes commitments to each share

   Every P_j obtains a share $[x_i]_j$ (from each P_i), as well its commitment $g^{[x_i]_j}$

   $[g^{x_{21}}, g^{x_{22}}, \dots, g^{x_{2n}}]$

2. Each P_j computes the sum of obtained shares $s_j = \sum_i [x_i]_j$ as well as g^{s_j}

   $g^{x_{11}} \cdot g^{x_{21}} = g^{x_{11}+x_{21}}$

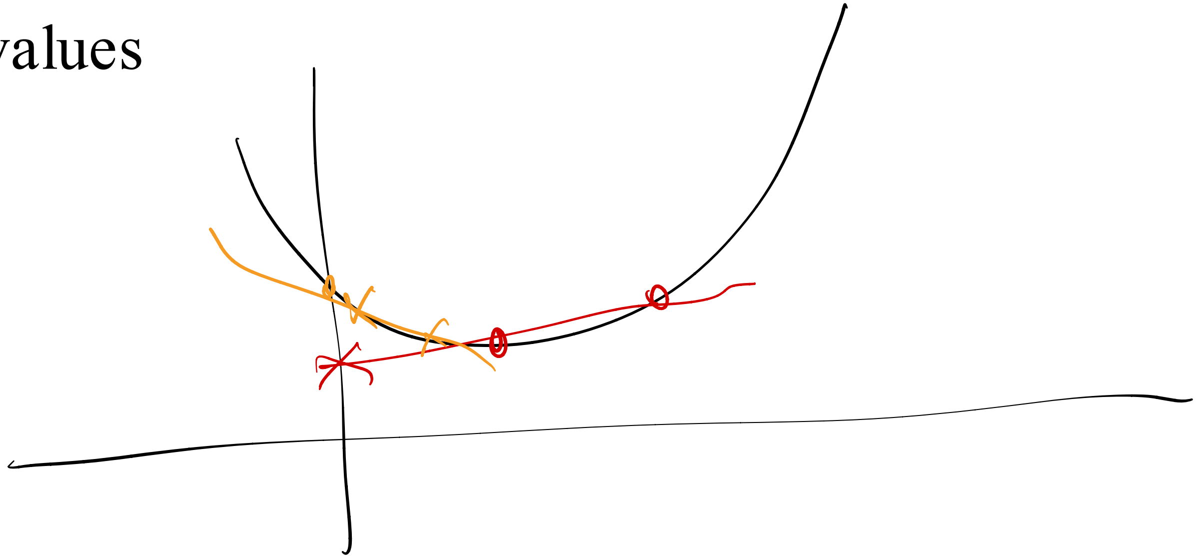
Note that here the values s_j lie on a degree-t polynomial, with secret x

The public key can be interpolated using the values $\{g^{s_j}\}_j$

Active Security

- Previous sketch does not work if dealer misbehaves
- Reason: Shamir-sharing does not guarantee binding if dealer is corrupted
  - Corrupted dealer can distribute shares on larger degree, and later reconstruct different values

$t=1$

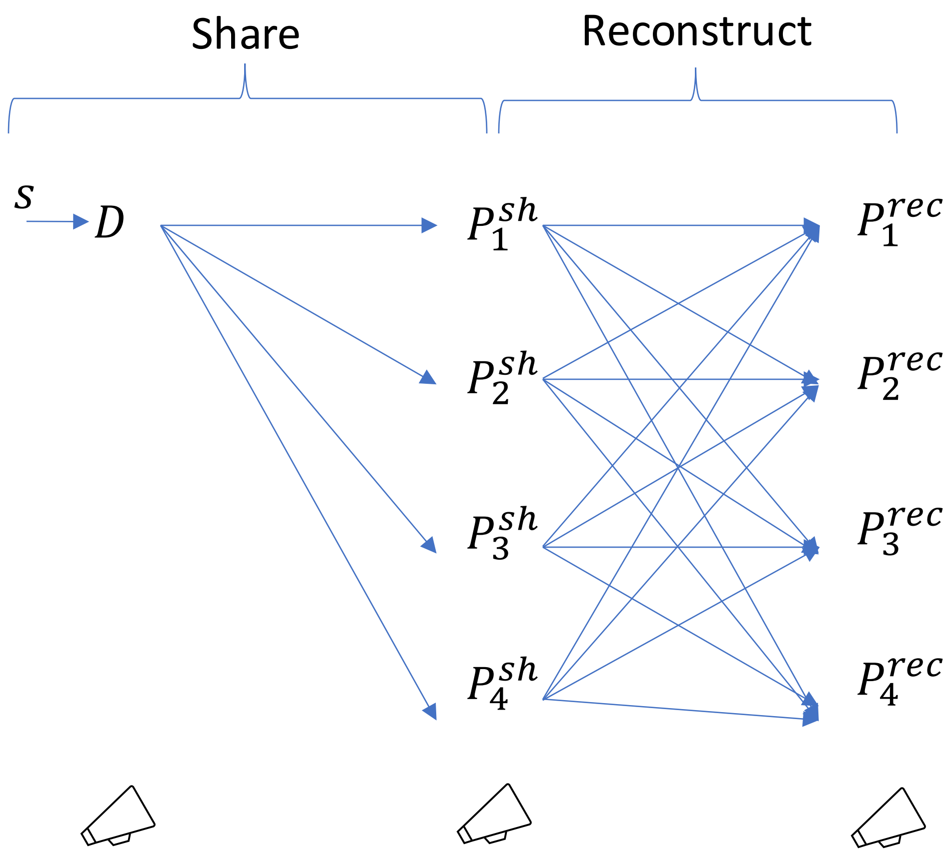


Active Security

- Previous sketch does not work if dealer misbehaves
- Reason: Shamir-sharing does not guarantee binding if dealer is corrupted
    - Corrupted dealer can distribute shares on larger degree, and later reconstruct different values

Fix: Verify that the shares lie on degree-t polynomial

Verifiable Secret Sharing



Commitment: After Share succeeds, only one value s' can be reconstructed.
And $s' = s$ if D is honest

Privacy: Secret not revealed during Share

Verifiable Secret Sharing

$s \longrightarrow D$

P_1 s_1

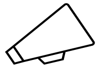
g^s
g^{f_1}
g^{f_2}
…
g^{f_t}

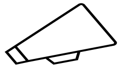
P_2 s_2

P_3 s_3

…

P_n s_n





$f(x) = s + f_1\, x + f_2 x^2 \ldots + f_t x^t$

$s_i \; = \; f(i)$

Verifiable Secret Sharing

g^{s_1}	g^{s_2}	g^{s_3}	...	$g^{s_{n-1}}$	g^{s_n}

$s \longrightarrow D$

g^{s}
g^{f_1}
g^{f_2}
...
g^{f_t}

P_1 s_1 ✓

P_2 $s_2{}'$ ✗

P_3 $s_3{}'$ ✗

...

P_n s_n ✓

$$f(x) = s + f_1\, x + f_2 x^2 \ldots + f_t x^t$$
$$s_i \;=\; f(i)$$

Verifiable Secret Sharing

g^{s_1}	g^{s_2}	g^{s_3}	…	$g^{s_{n-1}}$	g^{s_n}

$s \longrightarrow D$

g^{s}
g^{f_1}
g^{f_2}
…
g^{f_t}

P_1 s_1 ✓

P_2 ${s_2}'$ ✗ s_2

P_3 ${s_3}'$ ✗ s_3

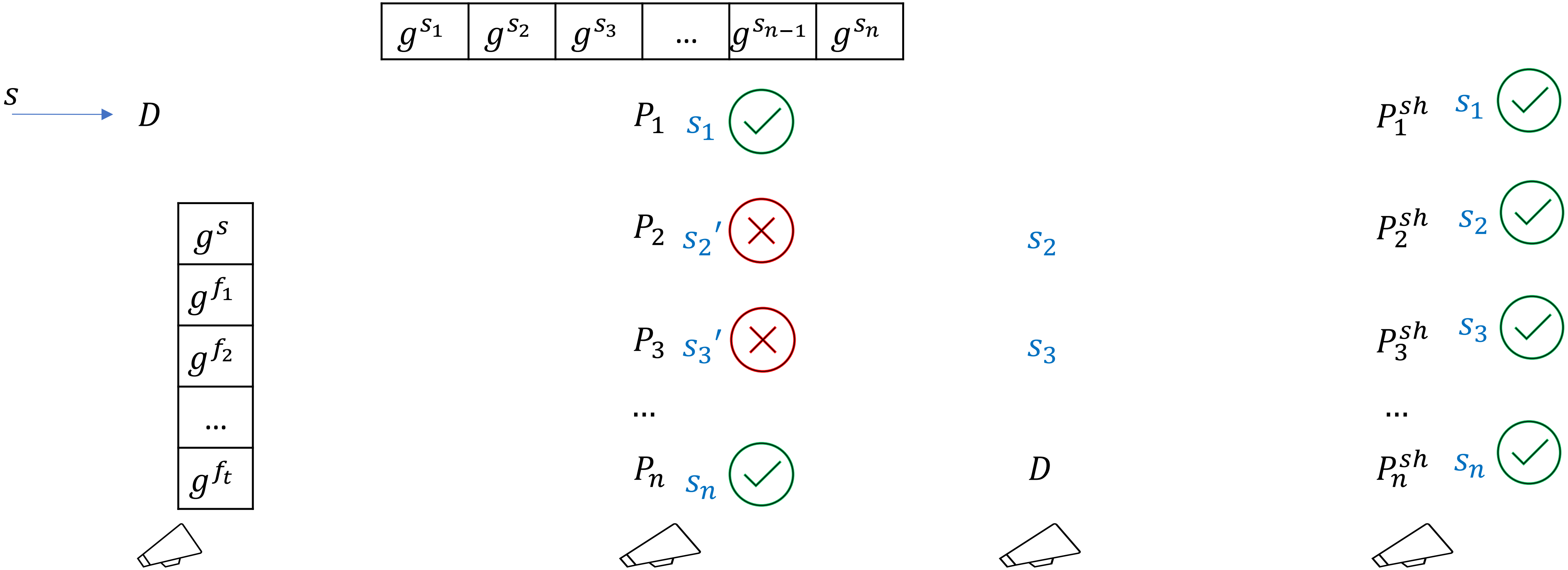
…

P_n s_n ✓ D

$$f(x) = s + f_1\, x + f_2 x^2 \ldots + f_t x^t$$

$$s_i \;=\; f(i)$$

Verifiable Secret Sharing



$f(x) = s + f_1\, x + f_2 x^2 \ldots + f_t x^t$

$s_i \;=\; f(i)$

Architecture

Distributed Key Generation

Distributed Protocol for the specific task: signing, decrypting, etc

Architecture

Distributed Key Generation

Distributed Protocol for the specific task: signing, decrypting, etc

Example: BLS Signature

Key generation: $sk = x, pk = g^x$

To sign a message m

- Compute signature $\sigma = (H(m))^x$

To verify a message m' and signature σ'

- Checked using pairings, that there is an x' such that $pk = g^{x'}$ and $\sigma' = (H(m'))^x$

Example: Threshold BLS Signature

Distributed Key generation: $sk = x$, $pk = g^x$

- Each honest P_i obtains a share s_i of the secret key sk, and everyone has commitment values $(g^{s_1}, \dots, g^{s_n})$, as well as pk

To sign a message m

- Each party P_i publishes $\sigma_i = (H(m))^{s_i}$ (partial signature)
- Full signature σ can be computed using $t + 1$ partial signatures

Conclusion

Threshold Cryptography

==

Distributed Key Generation +

Distributed Computation using the Keys